Universitat de Lleida

TREBALL FINAL DE GRAU

Estudiant: **Lucian Marian Nedelcu**

Titulació: Grau en Enginyeria Informàtica

Títol de Treball Final de Grau: **Desenvolupament d'una aplicació web per gestionar les actuacions en granges**

Director/a: **Roberto García Gonzalez i Joan Molló Solsona**

Presentació

Mes: Juny

Any: 2018

Resum

Aquest projecte té com finalitat desenvolupar un mòdul de gestió de tasques, treballant dins d'un context agrícola.

L'equip de comerç va descobrir que els grangers tenien molt d'interès en la planificació i organització del seu treball de dia a dia. Per donar una millor experiència es va decidir afegir aquesta nova funcionalitat a la eina que molts ja feien servir per gestionar les seves comandes.

Aquest document conté una descripció de tot el procés de creació del mòdul, raonant i explicant les decisions, les eines i el treball que s'ha fet per lliurar el millor programari.

Continguts

Llista de Figures

Abreviacions

FP - (Feed Planner), la solució multiplataforma per planificar el creixement d'animals gestionant comandes de pinso, predint la seva evolució i mes.

MVC - (Model View Controller), és un dels tres models de programació ASP.NET; també es un patró de disseny web. El model representa el nucli d'aplicació, la vista mostra les dades i el controlador que és l'encarregat de vincular-les.

FPW - (Feed Planner Web), la part del projecte Feed Planner que tracte amb la part de web.

FPD - (Feed Planner Desktop), la versió inicial de Feed Planner desenvolupada com aplicació d'escriptori.

FPM - (Feed Planner Mobile), part del projecte Feed Planner que engloba les apps per Android i iOS.

CRUD - (Create Read Update Delete) en programació són les funcions de crear, llegir, actualitzar i esborrar, les bàsiques per qualsevol funcionalitat d'emmagatzematge.

Enum - (Enumerator), en programació és un tipus de dades que consisteix en una serie d'elements amb nom i valor.

JS - (Javascript), un llenguatge de Script[1] que permet fer gairebé tot en una pagina web.

[1] Script és un programa interpretat, no compilat pensat per automatitzar l'execució de tasques realitzades normalment per un usuari

1 Introducció

Motivació

Durant el 2018 mentre l'estudiant Lucian Marian Nedelcu completava les pràctiques a l'empresa Caedis Solutions va sorgir una oportunitat. Un equip de comercials van contactar l'equip per exposar unes noves necessitats a implementar per una aplicació enfocada a les granges. Com la empresa ja porta diversos projectes dins d'aquest àmbit l'estudiant es va oferir per realitzar el projecte.

Un cop acabades les pràctiques l'empresa va decidir analitzar la aplicació i l'estudiant va començar a preparar-se per la tasca.

Objectius

L'objectiu principal es construir una aplicació web que gestiona les tasques a realitzar en diverses granges. L'enfoc de la aplicació seran les diferents persones que actuen en una granja com poden ser els granger mateix, els veterinaris, els visitants i altres.

També cal mantenir l'estil i claredat de l'aplicació germana: Feed Planner, que també esta enfocada en les granges, la producció animal i gestió de temes importants com ara les comandes.

Per si això fos poc l'aplicació ha de tenir un component web, el nucli d'aquest projecte, i un component mòbil que hauran de connectar-se i comunicar amb el mateix servei.

Estructura de la resta del document

El contingut restant del informe queda dividit de la següent forma:

I. **Metodologia i Tecnologies utilitzades:** secció sobre el projecte en general, les decisions inicials i més importants per l'equip. conte un resum i anàlisis de les eines utilitzades en la implementació i testeig de l'aplicació.

II. **Desenvolupament i Implementació:** bloc que conte la part inicial del desenvolupament i tracta d'explicar moltes decisions i treball realitzar abans de afrontar la part més carregada del projecte. explicació del treball realitzat, comentant les tasques, les revisions i el codi realitzat.

III. **Conclusions i Treball futur:** conte un anàlisis i reflexió personal sobre el treball realitzat. bloc en el que s'examina la trajectòria futura del projecte, coses a afegir i millorar.

2 Metodologia

Per desenvolupar aquest projecte tan l'estudiant com l'equip van seguir unes normes i pautes a l'hora de treballar i dirigir la trajectòria final del programa. Una de les primeres va ser utilitzar un mètode de de desenvolupament àgil (Agile en angles).

Metodologies Agile Scrum

El terme Agile·l· engloba diverses metodologies similars que comparteixen bona part de la mateixa filosofia. així com moltes de les mateixes característiques i pràctiques. Però des del punt de vista de la implementació, cadascun té la seva pròpia recepta de pràctiques, terminologia i tàctiques.

En la metodologia Scrum, que és la que s'aplica en part en aquest projecte hi ha 3 membres importants: el propietari, l'equip i el client. El propietari s'encarrega de consultar el client i formular les funcionalitats que l'equip implementarà. El client rebrà un producte després de que el propietari hagi validat la correcció i performança del treball realitzat.

Per garantir que l'equip de desenvolupadors està sempre enfocat cap a la mateixa meta les funcionalitats s'agrupen i organitzen en un registre anomenat Backlog. Les tasques a realitzar sempre han d'estar definides al Backlog i qualsevol dubte s'ha de resoldre abans de iniciar un esprint.

Esprints

Ja que ha sorgit abans un Esprint [2] és un període de temps determinat durant el qual s'ha de completar un treball específic i estar preparat per a la seva revisió. Primer el propietari i l'equip tenen una reunió i estableixen quines tasques s'ha de realitzar en un període curt, de una a tres setmanes. L'equip s'organitza i analitza cada tasca establint la seva prioritat i relació amb altres tasques com també l'esforç que suposaria un programador o un desenvolupador realitzar la tasca per si sol. Un cop és te prou feina per tot l'equip s'assignen les tasques a cada membre de l'equip i comença la fase d'implementació.

Tot i que no sol passar, es possible que durant aquesta fase surtin noves tasques o noves funcionalitats o fins i tot errors deguts a tasques anteriors. Aquestes s'afegeixen al backlog i si dona temps s'acaben en aquests esprint. Aquests problemes es solen evitar implementant testos a la hora d'implementar les funcionalitats i a la hora d'afegir la funcionalitat a tot el projecte. En el cas d'aquest projecte no s'utilitza la metodologia TDD, que es com s'anomena ja que el projecte inicial no va començar amb aquest principi i no seria factible aplicar-ho només en aquest modul.

Utilitzant Contractes de dades

Com que aquest projecte acabara sent una web consultant un servei cal mirar de treballar de la forma més flexible i àgil. La forma d'enfoscar la capa on es guarden les dades de la capa que veu l'usuari que utilitzarem és el Data Contract o Contracte de dades. Aquest és un acord formal entre un servei i un client que descriu de forma abstracta les dades que s'han d'intercanviar. Això fa que el servei pugui contestar a peticions d'una aplicació web o mòbil o potser un altre servei.

3 Tecnologies utilitzades

Per començar i acabar aquest projecte fan falte moltes eines i tecnologies dissenyades per facilitar la feina al programador. Tot seguit s'enumera la llista de programes importants utilitzats de forma directa i indirecta al projecte.

Visual Studio 2017

El programa principal utilitzat pel desenvolupament del modul i les aplicacions predecessores. És l'entorn de desenvolupament de Microsoft i s'utilitza per programar jocs, aplicacions financeres, pàgines web, serveis i més. També permet crear aplicacions per a dispositius mòbils i escriptori i publicar-les des de mateix entorn. Potser la característica més important de VS és que s'integra amb altres solucions per facilitar la feina del desenvolupador i accelerar i garantir el bon funcionament del programa.

.NET MVC

VS 2017 és la versió més recent i permet utilitzar tot el potencial del llenguatge c#. Una de les més interessants és el ASP .NET MVC. Ofereix una manera potent i rapida de crear pàgines web dinàmiques. Aquest tipus de projecte porta un disseny a base de patrons, un control total de la interacció de l'usuari i és pot enfocar cap al disseny impulsats pel testeig o TDD.

El patró principal que s'utilitza és el Model Vista Controlador. aquest patró divideix l'aplicació en 3 parts repartint una responsabilitat en cada. El model s'encarrega del comportament de les dades, el seu format i estructura. La vista és informació que veu l'usuari en pantalla i amb la que interactua. El controlador accepta les peticions de l'usuari i els converteix en ordres per la vista i el model.

A la hora de crea un projecte MVC sigui .NET o altres totes segueixen una estructura semblant partint l'aplicació en de 'App' que s'encarrega de arrancar i mantenir la web, una part de 'Scripts' o llibreries externes que el projecte utilitza i finalment la part del MVC configurada i aïllada de les altres. Cal dir que les vistes que utilitza MVC no són simple html es guarden amb una extensió csthml. Utilitzen una tecnologia anomenada Razor.

Razor és una sintaxi de marca per incrustar el codi basat en el servidor a pàgines web. La sintaxi Razor es compon de Razor markup, C # i HTML. Per exemple : ** @Username **

aquest tros de codi esta compost per un paràgraf, en html i a dins una variable de c#, sempre comencen per una @ seguit del nom de la variable o {} envoltant tot el codi a executar.

Plugins

VS 2017 incorpora molta funcionalitat per un entorn de desenvolupament però hi ha encara més que s'hi pot afegit aquests són complements o nuggets, com els anomena Microsoft. A continuació apareixen només els més importants per al projecte:

I. Telerik MVC

És un Framework[2] que es pot incorporar al projecte i es centra en la interfície. Serveix per a qualsevol escenari d'aplicació .NET. Redueix el temps de desenvolupament sense disminuir la qualitat gràfica o la funcionalitat de la solució. La UI[3] de Telerik® ofereix més de 60 components basats en jQuery obligatoris per a totes les aplicacions, desde quadrícules, menús, a controls avançats de línia de negoci, com ara gràfics, diagrames, calendaris fins a mapes.

II. Resx Manager

Serveix per localitzar i gestionar tot tipus de recursos basats en resx. Mostra tots els recursos i et permet editar les cadenes i les seves ubicacions en una xarxa de dades ben organitzada.
El format de fitxer de recursos .resx consisteix en entrades XML[4], que especifiquen objectes i cadenes dins de les etiquetes XML; Dit d'un altre forma permet guardar un recurs de text sota diferents interpretacions i per aquest motiu s'utilitza per traduir els recursos de text d'una aplicació.
Concretament en les aplicacions de Caedis tots els elements tenen una traducció anglesa i catalana.

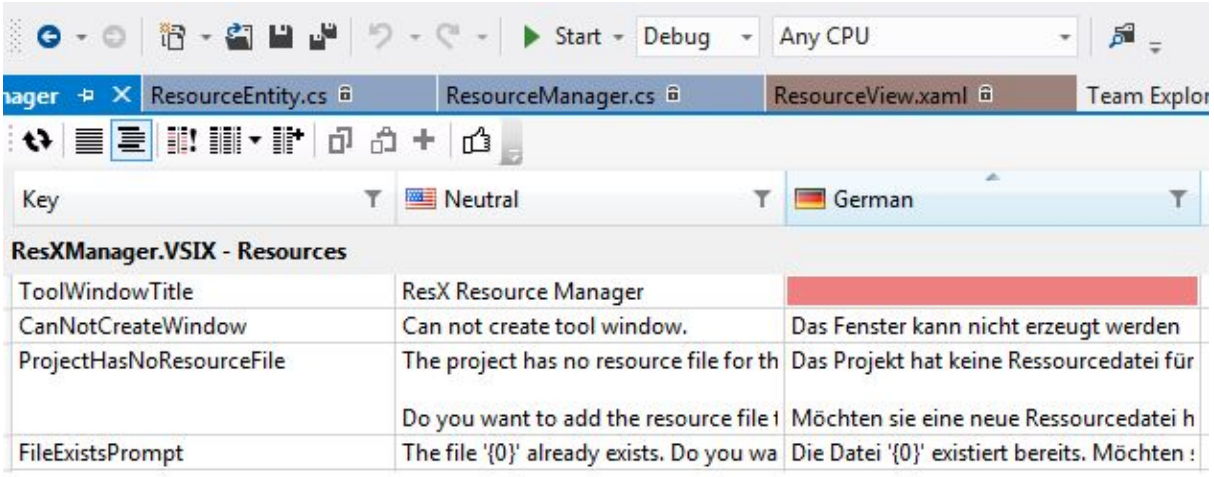

Figura 1: Fitxer obert amb ResX Manager

III. Tortoise SVN

És un client de subversions Apache™, implementat com una extensió del sistema operatiu de Windows. És fàcil d'utilitzar, ja que no requereix que el client extern per executar el control de codi.
El programa s'encarrega de generar i mantenir el codi del projecte incorporant totes les aportacions parcials dels membres de l'equip al projecte.

IV. Jquery

És una biblioteca ràpida i lleugera de JavaScript compatible amb la majoria de navegadors del mercat. El framework és extensible i maneja molt bé les manipulacions DOM, AJAX, animacions i més. Aquesta llibreria és disponible com un nugget per VS i s'utilitza també en relació amb els altres paquets.

[2] Framework és una aplicació genèrica que permet al programador desenvolupar aplicacions o funcionalitats
[3] UI és refereix a la interfície d'usuari
[4] XML és un meta-llenguatge que permet definir llenguatges de marques propis.

V. LinQ

Consulta Integrada al Llenguatge[5] és el nom d'un conjunt de tecnologies pensades en la integració de les capacitats de consulta de base de dades directament en el llenguatge C #. El raonament és de estalviar al programador tenir que aprendre un llenguatge per fer consultes com ara el SQL i treballar de forma transparent amb les classes que el llenguatge proporciona. Amb LINQ, una consulta és una construcció de llenguatge de primera classe, igual que classes, mètodes, esdeveniments.

SQL Server 2017

És el programa utilitzat per crear i gestionar la base de dades basat en SQL. El sistema està dissenyat per al seu ús en aplicacions corporatives, tant a nivell local com al núvol.

Gemini Tracker

És un programa de gestió de projectes basat en .NET i permet la creació i seguiment de tasques o assumptes; Incorpora unes regles pre-establertes que es poden modificar i ampliar, permet definir equips, esprints projectes i més. És una eina molt útil i necessària en el desenvolupament Agile d'aplicacions.

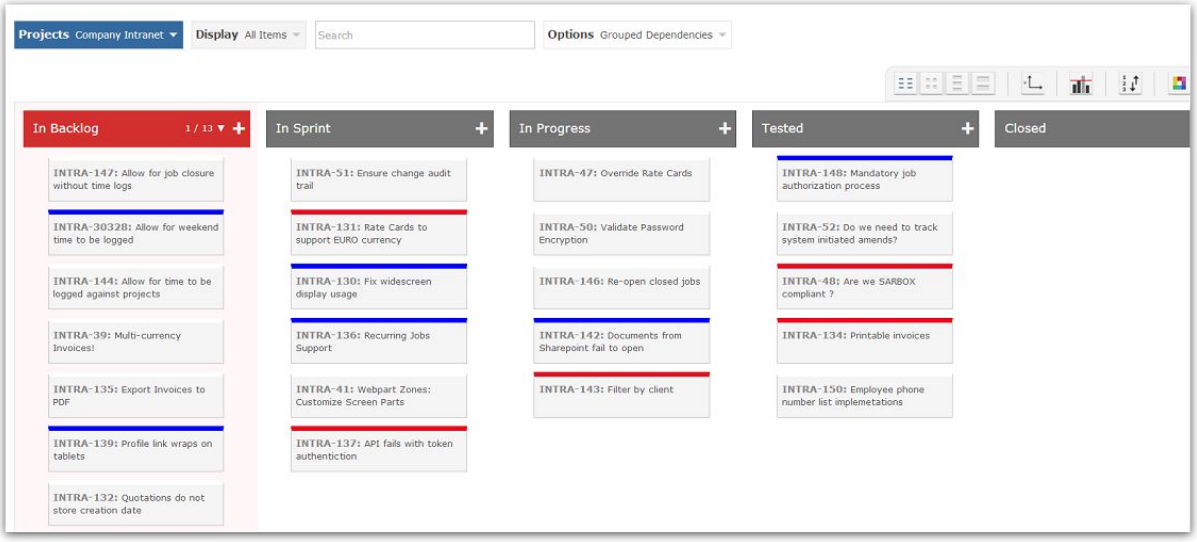

Figura 2: Organitzar tasques amb el Gemini Tracker

Chrome

És un navegador web gratuït pensat tan pels usuaris normals com per als desenvolupadors. A part d'oferir una bona experiència també incorpora moltes eines de testeig i depuració de pagines web. Disposa de un mode desenvolupador on es mostra el codi, les consultes, les respostes i possible errors en la aplicació. A més permet simular un dispositiu diferent per veure com s'adapta el disseny de la solució a diversos pantalles i mides per estar segur que tots els usuaris disposen de la millor experiència.

[5] Language Integrated Query en anglès

4 Desenvolupament

Per començar un projecte nou cal pensar i estudiar les lliçons dels projectes previs. Com que l'aplicació esta pensada a la mateixa gent que han comprat les solucions existents s'utilitzaran com una guia de disseny i treball.

Treball a seguir

L'aplicació FPW ja es arrencada i funcionant així que molts recursos i bones practiques s'han aplicat al projecte seguint-lo com a guia.

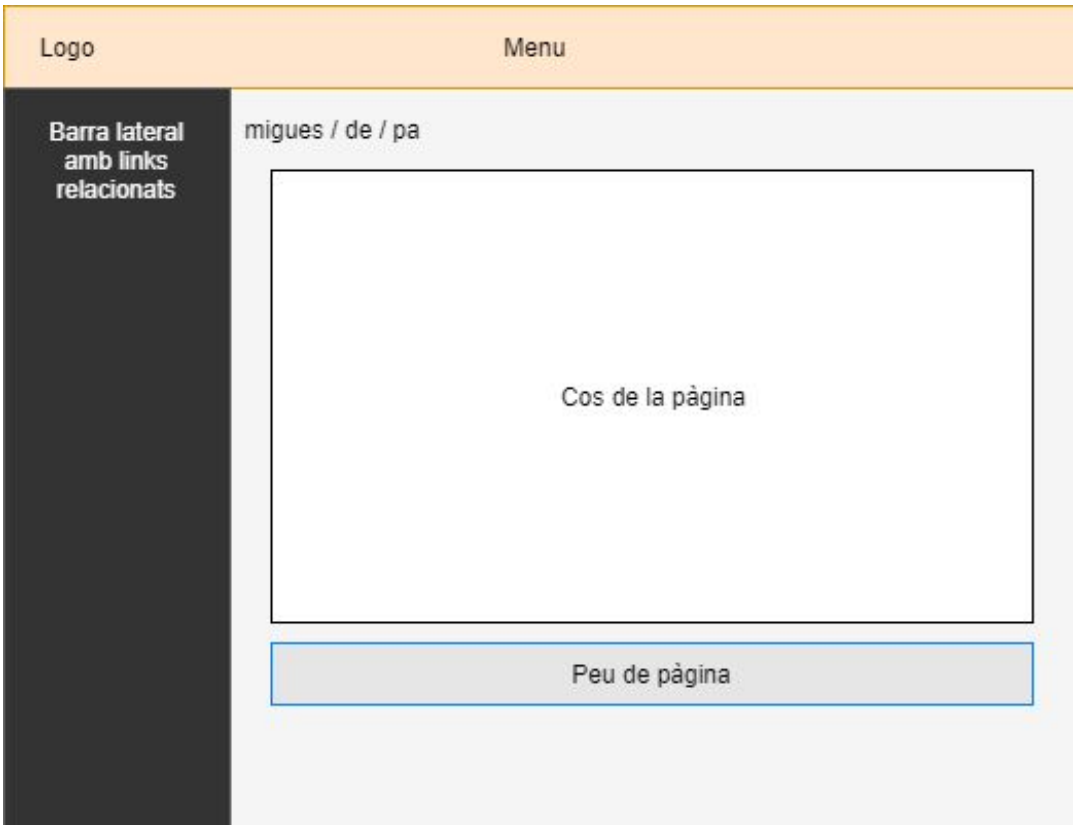

Figura 3: Plantilla de disseny de FPW

En termes de disseny la web ja conte una estructura definida, és multilingüe i és ***responsive***[6] i cal mantenir tota la nova funcionalitat sota els mateixos criteris.

La Barra lateral s'amaga i el Menú passa a ser una menú vertical en dispositius portables.

L'usuari pot canviar d'idioma en qualsevol moment i la pagina és tradueix segons les seves necessitats. Aquest son només uns dels criteris que el projecte haurà de satisfer per complir amb l'estandard existent dins de les solucions de l'empresa.

[6]responsive: l'aplicació web s'adapta el disseny per dispositius més petits com mòbils o portàtils.

Web o Addon[7] ?

La primera qüestió per resoldre és elegir la forma d'afrontar el projecte: com una aplicació independent o com un modul més de FP?

En aquest cas l'equip ha elegit treballar com si "Tasques" fos una funcionalitat addicional de la solució existent. Els motius són els següents:

I. Gran part del treball del servei ja esta fet, es podrien aprofitar classes, taules de la base de dades i lògica pre-establerta de domini.
II. Si el mòdul es tractes com una aplicació independent i s'implementes no es podria vendre sol ja que te massa dependències cap als usuaris de FP.

Anàlisi pel calendari

Una part critica de qualsevol gestor de tasques o events és el calendari ja que presenta molta informació de forma clara i estableix una funcionalitat estable i ampliable. Tot i que no s'han dissenyat les taules de la base de dades ni s'han creat els components segur que hi haurà un calendari tan a l'aplicació web com a la FPM.

Per aquest motiu cal buscar un que s'adapti a les necessitats del projecte. Cal fer un petit estudi dels components en oferta.

I. Full Calendar jQuery Plugin

Figura 4: Calendari FullCalendar

[7] Addon es refereix a un objecte afegit al ja existent que li dona més valor o utilitat.

Aquesta opció seria la més rapida i fàcil d'implementar ja que simplement caldria instal·lar el plugin i utilitzar-lo. En quan a la documentació hi ha diversos pagines que contenen exemples i tutorials sobre el calendari.

En quant a característiques té vistes de dia, més i agenda, opcions de localització, creació d'esdeveniments, diferents temes i la possibilitat d'afegir més funcionalitats amb javascript.

II. Essential JS 2

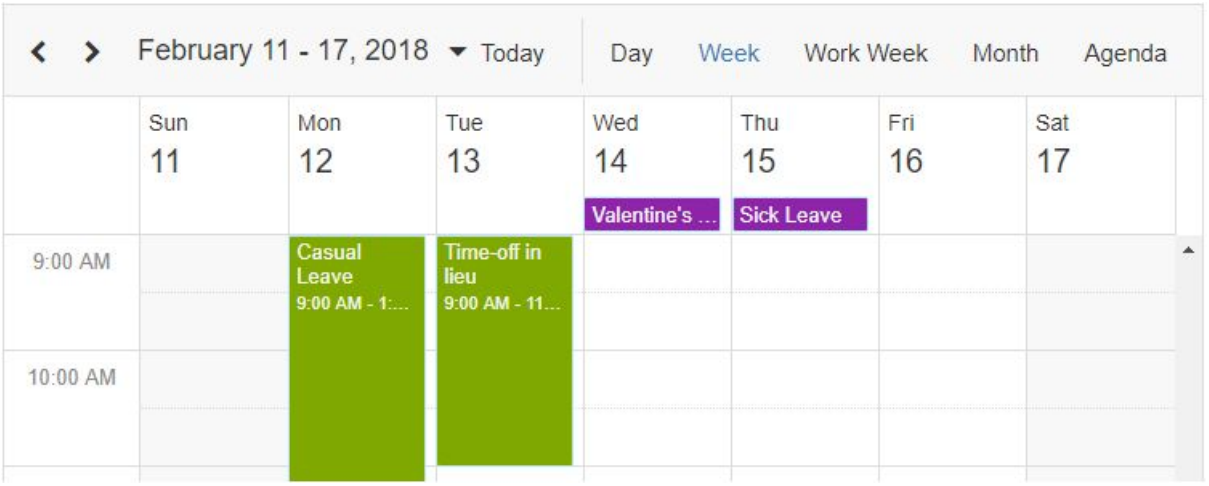

Figura 5: Calendari de Essentials JS 2

Un altre opció és utilitzar un paquet com el de Syncfusion. El seu kit: Essential JS 2 agrupa una serie de solucions per l'interfície en JavaScript i està pensat per ser lleuger, modular i orientat a aplicacions tàctils.

Entre els seus components incorpora una agenda que permet afegir modificar i esborrar esdeveniments. Proporciona diferents vistes i conte regles per events que es repeteixen. Dispose de diferents temes i validacions i també es pot operar amb el teclat.

III. iCalendar

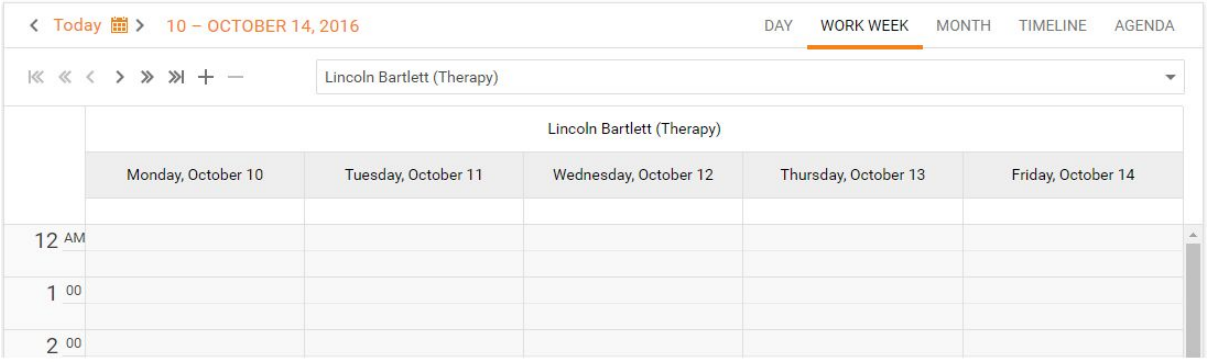

Figura 6: Calendari iCalendar

Un altre alternativa semblant és el component calendari dels de DevExpress. Funciona amb .Net MVc i conte totes les funcionalitats que un espera d'un calendari: diferents vistes, agrupació d'events per diversos usuaris, te un disseny adaptable a tots els dispositius, pot mostrar varies zones horàries i cal

mencionar el disseny més clar i optim. En canvi per la part de programació sembla ser el més complicat d'utilitzar i ofereix poca flexibilitat a l'hora d'adaptar el component a noves necessitats.

IV. Telerik MVC

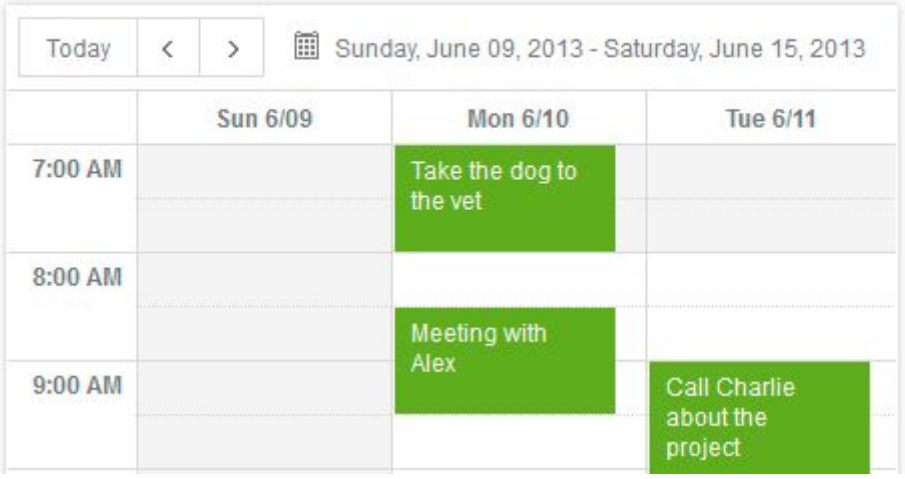

Figura 7: Calendari Telerik MVC Scheduler

La ultima opció a avaluar és el component Scheduler del paquet Telerik MVC. Ja que l'equip ha utilitzat diferents components semblants del Telerik aprendre a treballar amb aquest seria ràpid i no costaria res incorporar-lo a la web. en quan a funcionalitat el planificador disposa de diferents temes, vistes, opcions per crear i gestionar events, exportació a PDF i a més permet definir camps addicionals de forma simple. També permet personalitzar l'aparença de l'event dins del calendari.

Després d'avaluar tots els components cal establir quin s'utilitzarà al projecte i la resposta és simple: Telerik MVC Scheduler. El component més familiar per l'equip i també el component més ràpid d'implementar. A més cal dir que l'aplicació mòbil també fa servir els controls de telerik i que aquesta solució servira també a FPM.

A part d'aquests motius cal mencionar que les solucions 2 i 3 implicarien un cost de ≈ 900€ al projecte. Sense saber si s'utilitzaran cap dels altres components que contenen, no seria una bona inversió.

5 Implementació

Introducció a Feed Planner

Abans de parlar del codi del modul de tasques cal establir el seu projecte pare: FP i precisament FPW. Aquesta plataforma s'encarrega de gestionar granges, comandes, seguiment de creixement d'animals i en un futur gestionar tasques.

L'estructura del projecte web és un MVC estàndard: cada vista te un controlador que treballa amb un model de les dades rebudes de la base de dades. El projecte te una vista per defecte i principal: "_Layout.cshtml", serveix de pare o guia per totes les altres vistes de l'aplicació; Aquest fitxer conte els menús, els scripts i llibreries utilitzades com també el peu de pagina.

Primera reunió

Per començar el projecte i el primer esprint cal tenir una reunió de planificació. Aquesta reunió afecta a tot l'equip i per tant totes les aportacions son considerades.

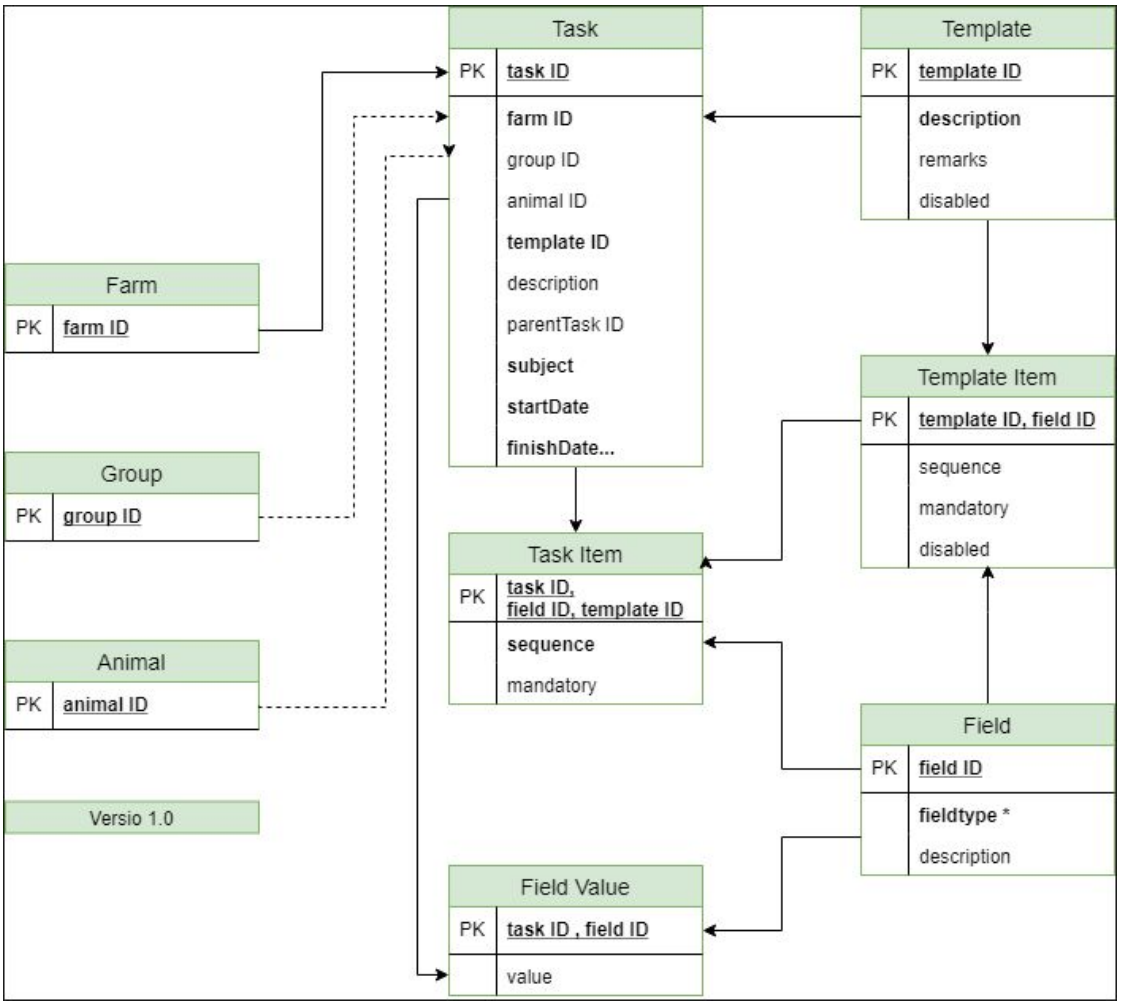

Figura 8: Esquema UML de la base de dades del projecte 1.0

Aquí es va decidir quina estructura de base de dades faríem servir i que el projecte s'acoblaria a FP com una nova funcionalitat.

Com la majoria de documents de l'empresa esta en anglès però es simple d'entendre; Hi ha 3 noves taules principals: tasca, plantilla i camp.

Un camp és un element, pot ser un nombre, una data o un text. La plantilla és un conjunt agrupat de camps per re-utilització fàcil. La tasca te una plantilla assignada i això fa que la tasca tingui els camps definits per la plantilla. Per completar la tasca cal modificar o emplenar els camps de la tasca. Aquests es guarden separat dels de la plantilla ja que estan pertanyen a una tasca.

Com que aquest document esta en català i la majoria de la informació del projecte en anglès cal precisar que:

Task - es refereix a la entitat tasca, **Task Item** - al camp de la tasca,

Template - la plantilla, **Template Item** - el camp assignat a la plantilla,

Field - és la entitat camp en general.

Esprint 1

Com a resultat de la planificació s'han establert les següents tasques a implementar:

Tasca	Funcionalitat	Estimació	Dependències
implementar la entitat"camp"	● llistar camps, ● afegir camp ● esborrar camp ● editar camp	10 hores	cap
implementar la entitat "plantilla de camps"	● llistar plantilles ● afegir plantilla ● esborrar plantilla ● editar plantilla ● afegir camps a una plantilla	8 hores	la plantilla esta formada per diversos "camps".
implementar la entitat "tasca"	● llistar tasques, ● afegir tasca ● esborrar tasca, ● editar tasca ● assignar una plantilla a una tasca	20+ hores	una tasca depèn de la "plantilla" assignada que fa que tingui 0 o diversos "camps".

Figura 9: Taula 1: Backlog de tasques de l'esprint 1

Tasca 1.1: Entitat Camp V0

Com que el projecte i les eines utilitzades estan pensades per utilitzar el model mvc partirem les entitats en 3 parts explicant que s'ha realitzat per al model la vista i el controlador.

El model

El primer pas es definir com serà la estructura que es mostrara en la web. Aquesta estructura depèn en part de la taula relacionada a la base de dades però també del comportament de la web.

```
public class TaskFieldModel
{
    4 references | 0 exceptions, ? live
    public int FieldId { get; set; }

    [UIHint ("FieldTypeEditor")]
    2 references | 0 exceptions, ? live
    public FieldType FieldType { get; set; }

    3 references | 0 exceptions, ? live
    public string Description { get; set; }
```

Figura 10: Snippet 1: Model de Field

El **FieldId** fa falta per identificar i distingir camps semblants.

El **FieldType** determina quin tipus de valor es guardara i quin control apareixerà en la web.

```
16 references
public enum FieldType
{
    Integer = 1,
    String = 2,
    Boolean = 3,
    Decimal = 4,
    DateTime = 5,
}
```

Figura 11: Snippet 2: Enum de FieldType

El **Description** serveix per descriure el camp, es com el títol del camp.

Per tal de comunicar-se amb el servei no hi ha prou en definir el model cal mapejar cada camp del contracte al corresponent del model assegurant una integritat entre els tipus.

Aquesta definició assegura que cada cop que obtenim un Camp del servei apareixerà com esta definit al model, però també cal afegint una relació en el sentit contrari per quan es vol guardar o enviar el model cap al servei.

```
public static implicit operator TaskFieldModel(Agrifood.CloudFP.Common.CloudFPService.FieldsContract x)
{
    TaskFieldModel fvm = new TaskFieldModel()
    {
        FieldId = x.FieldId,
        Description = x.Description,
        FieldType = (FieldType)x.FieldType,
    };
    return fvm;
}
```

Figura 12: Snippet 3: Funció mapejar contracte-model Field

```
public static implicit operator Agrifood.CloudFP.Common.CloudFPService.FieldsContract(TaskFieldModel x)
{
    Agrifood.CloudFP.Common.CloudFPService.FieldsContract fc = new Agrifood.CloudFP.Common.CloudFPService.FieldsContract()
    {
        FieldId = x.FieldId,
        Description = x.Description,
        FieldType = (int)x.FieldType,

    };

    return fc;
```

Figura 13: Snippet 4: Funció mapejar model-contracte Field

El controlador

El següent pas és utilitzar les dades proporcionades per mostrar, esborrar o editar i més. La primera funció s'encarrega de mostra una una vista simple.

```
public class FieldsController : Controller
{

    [AuthRequired]
    [CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
    0 references | 1 request, ? live | 0 exceptions, ? live
    public ActionResult Index()
    {
        return View();
    }
```

Figura 14: Snippet 5: Funció per la vista Field

Per tal de garantir la seguretat de l'aplicació hi ha 2 verificacions: una és si l'usuari està autenticat a la web i l'altre és per veure si l'usuari te accés al component de tasques. Com que el modul de tasques és un afegit a FPW es pot veure com un servei opcional que el client decideix si vol contractar o no. Tots els usuaris que volen crear o gestionar alguna part de tasques hauran de contractar aquest component.

Amb això no hi ha prou per mostrar el llistat cal demanar al servei tots els camps i guardar-los al model i enviar-los a la vista. Per tal d'agilitzar el procés la funció serà asíncrona.

```
public async Task<ActionResult> ReadFields([DataSourceRequest]DataSourceRequest request)
{
  List<Models.TaskFieldModel> tfm = new List<Models.TaskFieldModel>();

  Agrifood.CloudFP.Common.CloudFPService.CloudFPServiceClient con = new Common.CloudFPService.CloudFPServiceClient();
  String token = Helpers.Auth.GetSessionToken(System.Web.HttpContext.Current.Request);

  try
  {
    var templist = await con.GetFieldsAsync(token, AppGlobals.APPLICATION_CODE);

    if (templist == null)
    {
      tfm = new List<Models.TaskFieldModel>();
    }
    else
    {
      tfm = templist.ToList().ConvertAll(s => (Models.TaskFieldModel)s);
    }
  }
  catch (Exception ex)
  {
    throw ex;
  }

  return Json(tfm.ToDataSourceResult(request), JsonRequestBehavior.AllowGet);
}
```

Figura 15: Snippet 6: Funció per obtenir tots els Camps

En aquesta funció s'utilitza per primer cop el model. Ja que el servei ens tornara un llistat cal declarar primer un llistat buit. Tot seguit cal intentar fer la consulta al servei identificat l'aplicació i l'usuari amb el token[8]. Si el servei no retorna cap camp no cal mostrar res, però si hi retorna una serie de camps cal guardar-los al model.

Abans de guardar el model cal convertir el vector que ens envia el servei a una llista i després convertir cada un dels objectes al model del Camp. Si hi ha cap excepció es captura i sinó s'envia el llistat encapsulat en format Json[9].

Cal dir que aquest enfocament no és habitual en MVC, sinó que normalment cada controlador fa servir la mateixa funció per retornar la vista del llistat amb les dades. En aquest cas es fa servir una forma diferent ja que les funcions de llegir, afegir,editar i esborrar s'apliquen sobre un element de la vista no sobre tota la vista. Aquesta element és el llistat de Camps creat amb l'ajuda del plugin de Telerik MVC que és el que també marca la pauta de anomenar la funció d'obtenir com "read[Classe]".

```
public ActionResult AddField([DataSourceRequest] DataSourceRequest request, Models.TaskFieldModel NewField)
{
  Agrifood.CloudFP.Common.CloudFPService.CloudFPServiceClient con = new Common.CloudFPService.CloudFPServiceClient();
  String token = Helpers.Auth.GetSessionToken(System.Web.HttpContext.Current.Request);

  if (NewField != null && ModelState.IsValid)
  {
    con.InsertField((Agrifood.CloudFP.Common.CloudFPService.FieldsContract)NewField,
                                          token, AppGlobals.APPLICATION_CODE);
  }

  return Json(new[] { NewField }.ToDataSourceResult(request));
}
```

Figura 16: Snippet 7: Funció d'afegir de Field

[8] token és un codi que el servidor proporciona als usuaris un cop han entrat a l'aplicació web
[9] Json és la forma més habitual que els navegadors envien i reben dades del servidor

La següent funció és la de afegir que també treballar directament amb l'element de telerik. En aquest cas la web crida la funció passant-li les dades introduïdes per l'usuari i després d'una verificació de validesa s'insereixen a la base de dades. Aquí és on es fa servir la segona funció del model per transformar-lo al contracte per inserir.

```
0 references | 0 requests, ? live | 0 exceptions, ? live
public ActionResult EditField([DataSourceRequest] DataSourceRequest request, Models.TaskFieldModel Field)
{
  Agrifood.CloudFP.Common.CloudFPService.CloudFPServiceClient con = new Common.CloudFPService.CloudFPServiceClient();
  String token = Helpers.Auth.GetSessionToken(System.Web.HttpContext.Current.Request);

  if (Field != null && ModelState.IsValid)
  {
    con.UpdateField((Agrifood.CloudFP.Common.CloudFPService.FieldsContract)Field,
                                          token, AppGlobals.APPLICATION_CODE);
  }

  return Json(new[] { Field }.ToDataSourceResult(request));
}
```

Figura 17: Snippet 8: Funció d'edició de Field

Després d'afegir cal declarar la funció per modificar el Camp. Aquesta funció és molt semblant a l'anterior l'únic canvi sent que treballa sobre un Camp ja existent i fa servir el BackEnd per actualitzar les dades.

```
public ActionResult DeleteField([DataSourceRequest] DataSourceRequest request, Models.TaskFieldModel Field)
{
  Agrifood.CloudFP.Common.CloudFPService.CloudFPServiceClient con = new Common.CloudFPService.CloudFPServiceClient();
  String token = Helpers.Auth.GetSessionToken(System.Web.HttpContext.Current.Request);

  if (Field != null && ModelState.IsValid)
  {
    var err = con.DeleteField(Field.FieldId, token, AppGlobals.APPLICATION_CODE);

    if (err.ErrorType == Common.CloudFPService.ErrorType.None)
    {
      return Json(new[] { "" }.ToDataSourceResult(request), JsonRequestBehavior.AllowGet);
    }
    else
    {
      ModelState.AddModelError("Error", Enum.GetName(typeof(Common.CloudFPService.ErrorType), err.ErrorType));
      return Json(new[] { Field }.ToDataSourceResult(request, ModelState));
    }

  }

  ModelState.AddModelError("Error", "Null");
  return Json(new[] { Field }.ToDataSourceResult(request, ModelState));
}
```

Figura 18: Snippet 9: Funció d'esborrar Field

L'ultima funció per acabar el controlador és la d'esborrar; Aquesta també fa la comprovació de validesa i a més espera una resposta del servidor. Si hi ha hagut cap error com per exemple: intentar esborrar un camp que no existeix a la base de dades s'informaria. Com que la majoria d'errors solen ser humans l'equip ha tipificat els tipus més freqüents d'errors que poden passar i aquesta informació es passa a l'usuari per informar si realment s'ha eliminat el camp o perquè no s'ha pogut fer.

Ara que ja tenim declarades les parts funcionals cal definir que i com ho veurà l'usuari a la vista. això comença creant un fitxer Index.chtml[10].

[10] El nom és el mateix que el de la primera funció del controlador ja que aquella retorna aquesta vista.

La vista

L'element principal de la vista és una taula que contindrà les dades i les accions implementades al controlador.

Figura 19: Estructura bàsica de la taula en la vista

```
@(Html.Kendo().Grid<Agrifood.CloudFP.WebApp.Models.TaskFieldModel>()
    .Name("Fields")
    .ToolBar(toolbar =>
    {
      toolbar.Create().Text(Agrifood.CloudFP.WebApp.Resources.TextResources.Global_btnAdd_Text);
    })
```

Figura 20: Snippet 10: Declaració de la taula de Camps

Primer cal declarar la taula, especificant que utilitzara el model declarat en les seves operacions. Ja que Telerik fa servir molt javascript per darrere cal donar un nom a tots els elements Kendo perquè ho farà servir com ID.

Després afegim una barra d'opcions amb la opció de crear o afegir. Aquest recurs apareixerà diferent depenent de la idioma seleccionada per l'usuari.

```
.Columns(columns =>
{
  columns.Command(command =>
  {
    command.Edit().Text(" ")
           .UpdateText(" ")
           .CancelText(" ");
    command.Custom("customDelete")
           .Click("customDelete")
           .Text(string.Format(@"<span class=""{0}""></span>", Agrifood.CloudFP.WebApp.Classes.AppGlobals.GridControlColumnDeleteButton));
  }).Width(Agrifood.CloudFP.WebApp.Classes.AppGlobals.GridControlColumnWithTwoButtons);
```

Figura 21: Snippet 11: Codi per eliminar i editar Field de la taula

A dins de la taula comencem a definir les columnes. La primera tindrà les opcions de modificar i esborrar el registre de la fila que pertany. En el cas de la segona funcionalitat afegim una crida a una funció de javascript que demana a l'usuari confirmar que realment vol esborrar el registre. Per no embrutar el codi s'hi posa només el nom de la funció i la implementació es farà a sota.

```
function customDelete(e) {
    e.preventDefault();
    var tr = $(e.target).closest("tr");
    var data = this.dataItem(tr);
    SelectedDataItem = data;

    var dialog = $("#dialog");
    dialog.data("kendoDialog").open();
}
function onConfirmmDelete(e) {
  var data = SelectedDataItem;
  $("#Fields").data("kendoGrid").dataSource.remove(data);
  $("#Fields").data("kendoGrid").dataSource.sync();
  $("#Fields").data("kendoGrid").dataSource.read();
}
function onDataSourceError(e) {
  console.log(e.errors);
  var dialog = $("#delete_error");
  dialog.data("kendoDialog").open();
}
```

Figura 22: Snippet 12: Funcions JS per eliminar Field

La primera funció prevé la crida directa de la funció d'eliminar del controlador. Després selecciona i guarda el Camp que pertany a la fila d'on s'ha cridat aquesta funció i mostra un dialog[11] de confirmació.

```
@(Html.Kendo().Dialog()
        .Name("dialog")
        .Title(Agrifood.CloudFP.WebApp.Resources.TextResources.Global_DeleteConfirmation_Title)
        .Closable(false)
        .Content(@Agrifood.CloudFP.WebApp.Resources.TextResources.Global_DeleteConfirmation_Text)
        .Modal(true)
        .Visible(false)
        .Actions(actions =>
        {
          actions.Add().Text(@Agrifood.CloudFP.WebApp.Resources.TextResources.Global_Yes_Text).Action("onConfirmmDelete");
          actions.Add().Text(@Agrifood.CloudFP.WebApp.Resources.TextResources.Cancel_Button).Primary(true);
        })
)
```

Figura 23: Snippet 13: Codi del Dialog de confirmació

El Model que apareix te un missatge preguntat a l'usuari si vol esborrar aquest element i dona 2 opcions: cancel·lar o seguir amb l'esborrat. Les dos opcions tanquem el dialog i la segona crida la funció que intenta treure la fila de la font de dades de la taula. Un cop s'ha tret s'aplica aquest canvi sobre la taula que fa que la taula cridi la funció del controlador d'eliminar. Si aquesta petició retorna un error es captura i es mostra un altre dialog amb un missatge d'error.

Per comprovar de que les dades realment s'han esborrat de la base de dades tornem a carregar la font de dades que crida a la funció de lectura del controlador.

[11] Dialog és un tipus de modal que apareix per sobre de la vista actual i presenta 1 o més accions

```
    columns.Bound(p => p.FieldId).Visible(false);
    columns.Bound(p => p.Description).Title(Agrifood.CloudFP.WebApp.Resources.TextResources.FieldDetails
    columns.Bound(p => p.FieldType).ClientTemplate("#= parseType(FieldType) #")
    .Title(Agrifood.CloudFP.WebApp.Resources.TextResources.TemplateFields_Name_Text);
})
.Filterable()
.Sortable()
```

Figura 24: Snippet 14: Definició de les columnes de la taula Field

Havent creat tota la part de gestió dels camps cal afegir les altre columnes pertinents al model implementat. L'identificador no es mostra però si en cap moment es vulgues ensenyar és qüestió de canviar una variable.

Aquesta vista esta preparada per oferir totes les opcions CRUD sobre la entitat Camp. A la hora d'editar els membres del model Telerik ofereix un mode d'edició per fila. Aquesta funciona transformat cada cel·la en un editor segons el tipus; Per exemple una variable booleana apareixeria com un checkbox[12]. Com que el tipus és un Enum cal tractar-lo de forma especial, cal treballar amb el valor però mostrar a l'usuari el nom de l'element en qüestió.

```
function parseType(e) {
    var id = parseInt(e)-1;
    var enums = @Html.Raw(states);

    return enums[id];
}
```

Figura 25: Snippet 15: Funció per parsejar el tipus de Field

Quan la taula llegeix guarda un nombre per identificar el tipus aquest es transforma mitjançant una funció JS. Però això no serveix per la part d'edició així que cal definir un control especial per aquesta funcionalitat.

```
@(Html.Kendo().DropDownList()
            .Name("FieldType")
            .BindTo(new SelectList(Enum.GetValues(typeof(Agrifood.CloudFP.WebApp.Classes.AppGlobals.FieldType))
                    .Cast<Agrifood.CloudFP.WebApp.Classes.AppGlobals.FieldType>()
                    .Select(v => new SelectListItem
                    {
                      Text = v.ToString(),
                      Value = ((int)v).ToString()
                    }).ToList(), "Value", "Text"))
)
```

Figura 26: Snippet 16: Vista per editar el tipus de Field

Aquest nou editor esta en un fitxer anomenat FieldTypeEditor.cshtml; Això és tot el que cal fer ja que abans en el model ja s'havia posat una anotació amb el nom d'aquesta vista parcial. Aquesta no és una coincidència ja que d'aquesta forma està relacionat el tipus amb la vista i es pot utilitzar en qualsevol vista on es vulgui editar o modificar.

Això és tot per Camp ara cal explicar la Plantilla.

[12] Checkbox és un tipus d'entrada de dades de formularis web amb 2 estats: comprovat i no comprovat

Tasca 1.2: Entitat Plantilla V0

Aquesta entitat funciona com una col·lecció de camps que s'assignaran a una tasca.

El model

Seguint l'exmple de Camp el primer a definir és l'estructura de dades de la plantilla.

```
public class TaskTemplateModel
{
    12 references | 0 exceptions, ? live
    public int TaskTemplateId { get; set; }

    9 references | 0 exceptions, ? live
    public string Description { get; set; }

    4 references | 0 exceptions, ? live
    public bool IsDisabled { get; set; }

    4 references | 0 exceptions, ? live
    public string Remarks { get; set; }
```

Figura 27: Snippet 17: Model de Template

El **TaskTemplateId** fa falta per identificar i distingir les plantilles.

El **Description** és el nom de la plantilla.

El **IsDisabled** és un camp que normalment serà fals, serveix per desactivar una plantilla sense esborrar-la; Ja que per exemple si hi hagués una plantilla assignada a varies tasques i s'elimines la plantilla de la base de dades, aquesta quedaria en un estat inconsistent; Millor optar per desactivar la plantilla.

El **Remarks** serveix com un camp suplementari per qualsevol necessitat de l'usuari.

```
public static implicit operator TaskTemplateModel(Agrifood.CloudFP.Common.CloudFPService.TaskTemplatesContract x)
{
    TaskTemplateModel tvm = new TaskTemplateModel()
    {
        TaskTemplateId = x.TaskTemplateId,
        Description = x.Description,
        IsDisabled = x.IsDisabled,
        Remarks = x.Remarks
    };

    return tvm;
}
```

Figura 28: Snippet 18: Funció mapejar model-contracte Template

Per consultar el servei definim un parsejador del contracte igual que abans en els 2 sentits: model - contracte i contracte-model. I el model queda fet, ara cal utilitzar-lo al controlador.

El controlador

Per no tornar a ensenyar i comentar el mateix codi només cal dir que les funcions: Index, Delete, Read són iguals que les de Camps canviant només el model i la funció del servei.

Havent dit això cal parlar de la funció d'afegir una plantilla ja que no es fa de la mateixa forma que el camp.

```
[AuthRequired]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 1 request, ? live | 0 exceptions, ? live
public ActionResult AddNewTemplate()
{
    TaskTemplateModel temp = new TaskTemplateModel()
    {
        TaskTemplateId = -1,
        Description = " ",
    };

    return View(temp);
}
```

Figura 29: Snippet 19: Funció de la vista de creació de Template

Aquesta funció no retorna un Json sinó que retorna una vista, amb el mateix nom que la funció. A més la vista es precàrrega amb el les dades del model assignant un ID i una descripció buida. Aquesta valors no són importants només serveixen per a que a la següent funció passi el test de validesa del model.

```
[AuthRequired]
[HttpPost]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 1 request, ? live | 0 exceptions, ? live
public ActionResult AddNewTemplate(TaskTemplateModel model)
{
    if (model != null && ModelState.IsValid)
    {
        Agrifood.CloudFP.Common.CloudFPService.CloudFPServiceClient con = new Common.CloudFPService.CloudFPServiceClient();
        String token = Helpers.Auth.GetSessionToken(System.Web.HttpContext.Current.Request);

        model.TaskTemplateId = con.InsertTaskTemplate((Agrifood.CloudFP.Common.CloudFPService.TaskTemplatesContract)model, token,
    }

    return RedirectToAction("Details", "TaskTemplates", new { TemplateId = model.TaskTemplateId });
}
```

Figura 30: Snippet 20: Funció de resposta de la vista de creació de Template

amb la funció anterior no en tenim prou per guardar una plantilla cal afegir una funció diferent que reacciona a una publicació [13]. Les dades publicades es converteixen al model, es validen i després s'insereixen al Backend. Aquesta funció retorna un Id amb el qual podem trobar les dades de la plantilla. Així que un cop afegida la plantilla s'envia a l'usuari cap a la pagina de detalls de la plantilla que creada.

[13] Publicació, POST en anglès és un del tipus bàsics de peticions http que es poden fer a un servidor.

```
public async Task<ActionResult> Details(int TemplateId)
{
  Models.TaskTemplateModel Details = new Models.TaskTemplateModel();

  Agrifood.CloudFP.Common.CloudFPService.CloudFPServiceClient con = new Common.CloudFPService.CloudFPServiceClient();
  String token = Helpers.Auth.GetSessionToken(System.Web.HttpContext.Current.Request);

  try
  {
    if (TemplateId != -1)
    {
      Details = await con.GetTaskTemplateAsync(TemplateId, token, AppGlobals.APPLICATION_CODE);

      if (Details.TaskTemplateId.Equals(0))
        ViewBag.NoDetails = true;
      else
        ViewBag.NoDetails = false;
    }
  }
  catch (Exception ex)
  {
    throw ex;
  }

  return View(Details);
}
```

Figura 31: Snippet 21: Funció de la vista d'edició de Template

La vista de detalls funció de la mateixa forma. Hi ha una funció que torna la vista i una que espera un formulari complert amb les dades modificades de la plantilla.

Per veure i modifica una plantilla primer cal passar-li un ID; Si aquest ID no es valid o la plantilla no existeix la vista tindrà un model buit i mostrara un error. Aquí s'utilitza un altre forma de passar les dades cap a la vista. Apart del model en .NET MVC es poden guardar dades en un ViewBag que ve a ser com un secret entre la vista i el controlador ja que independent del model i és confidencial, és a dir cada parell vista-controlador utilitza un ViewBag diferent. En aquest cas només es guarda una variable d'error.

```
public ActionResult Details(TaskTemplateModel model)
{
  if (model != null && ModelState.IsValid)
  {
    Agrifood.CloudFP.Common.CloudFPService.CloudFPServiceClient con = new Common.CloudFPService.CloudFPServiceClient();
    String token = Helpers.Auth.GetSessionToken(System.Web.HttpContext.Current.Request);

    con.UpdateTaskTemplate((Agrifood.CloudFP.Common.CloudFPService.TaskTemplatesContract)model, token, AppGlobals.APPLI
  }

  return RedirectToAction("Details", "TaskTemplates", new { TemplateId = model.TaskTemplateId });
}
```

Figura 32: Snippet 22: Funció de resposta de la vista d'edició de Template

De la mateixa forma la vista de detalls te una funció de publicació que rep el model, el valida i actualitza. Un cop actualitzades les dades la funció envia a l'usuari cap la vista de detalls ja actualitzada. Tot i que per l'usuari sembla que esta recarregant la pagina per darrera el controlador tracte les 2 funcions diferent i si realment mostres la mateixa vista amb les noves dades no hi hauria cap garantia des del servei de que els canvis s'han produït.

La vista

La plantilla es diferent al Camp ja que conté 3 vistes enlloc d'una. Aquestes vistes son:

I. La vista del llistat

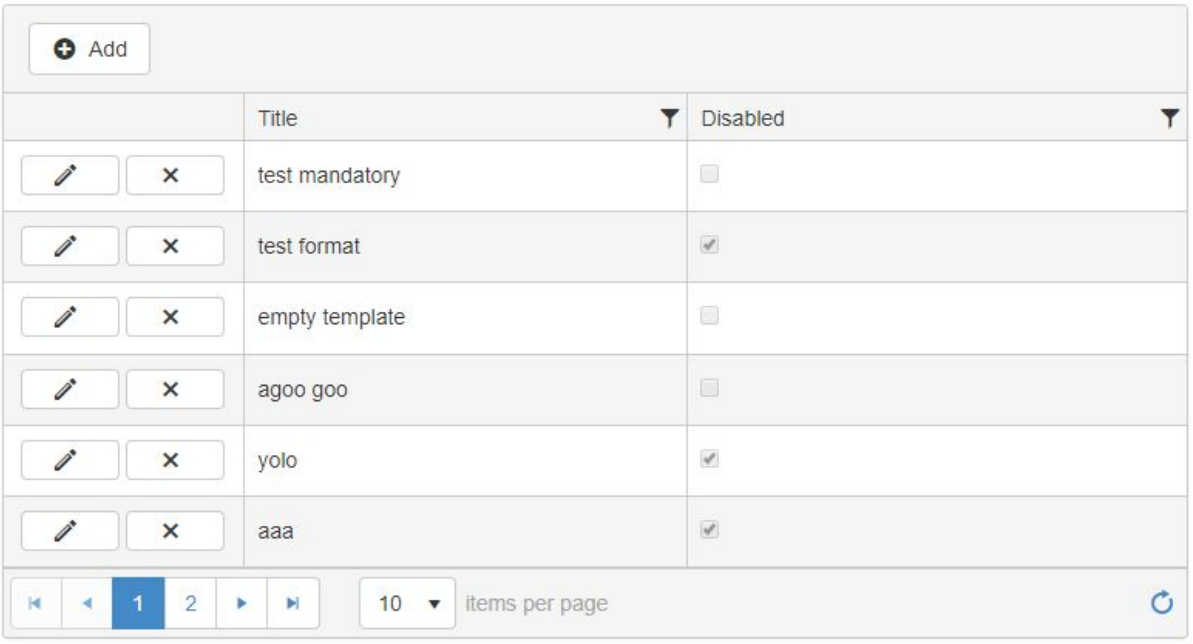

Figura 33: Vista llista de de Template

Per tal de llistar els elements fem servir el mateix control de Telerik el Kendo.Grid amb el model de la plantilla.

```
toolbar.Template(
  @<text>
    <button type="button" id="AddTemplate" class="k-button k-state-default panel-edit-button">
  </text>);
})
```

Figura 34: Snippet 23: Codi del botó d'afegir de la vista llista

```
$('#AddTemplate').click(function () {
  window.location.href = "/TaskTemplates/AddNewTemplate";
})
```

Figura 35: Snippet 24: Funció JS per cridar la funció d'afegir Template

A la barra d'opcions de la taula afegim un botó per crear una plantilla però com que es vol que això porti cap a un altre pagina no s'utilitza un mètode per defecte sinó que dins del codi c# de la taula afegim codi html per mostrar un botó amb estil semblant al que crear la taula per defecte. Aquest boto es configura a sota per redirigir l'usuari cap a la vista d'afegir creada al controlador.

```
.DataSource(dataSource => dataSource
  .Ajax()
    .Model(model =>
    {
      model.Id(p => p.TaskTemplateId);
    })
  .Events(e =>e.Error("onDataSourceError"))
  .Read(read => read.Action("ReadTemplates", "TaskTemplates"))
  .Destroy(destroy => destroy.Action("DeleteTemplate", "TaskTemplates"))
)
```

Figura 36: Snippet 25: Codi per configurar la font de dades de la taula

Això també implica canviar la font de dades, ja que només cal establir quines funciones es faran servir per carregar i esborrar les plantilles. I el tractament a la hora d'esborrar elements és el mateix que amb Camps.

```
command.Custom("showDetails")
        .Click("showDetails")
        .Text(string.Format(@"<span class=""{0}""></span>",
command.Custom("customDelete")
       .Click("customDelete")
       .Text(string.Format(@"<span class=""{0}""></span>", A
```

Figura 37: Snippet 26: Codi per eliminar i editar Template de la taula

A la hora de modificar la plantilla l'usuari selecciona la mateixa opció que abans però ara aquest boto el porta cap a un altre pagina.

```
function showDetails(e) {
  var dataItem = this.dataItem($(e.currentTarget).closest("tr"));
  window.location.href = "/TaskTemplates/Details/?TemplateId=" + dataItem.TaskTemplateId;
}
```

Figura 38: Snippet 27: Funció JS per cridar la funció d'edició de Template

II. La vista per afegir

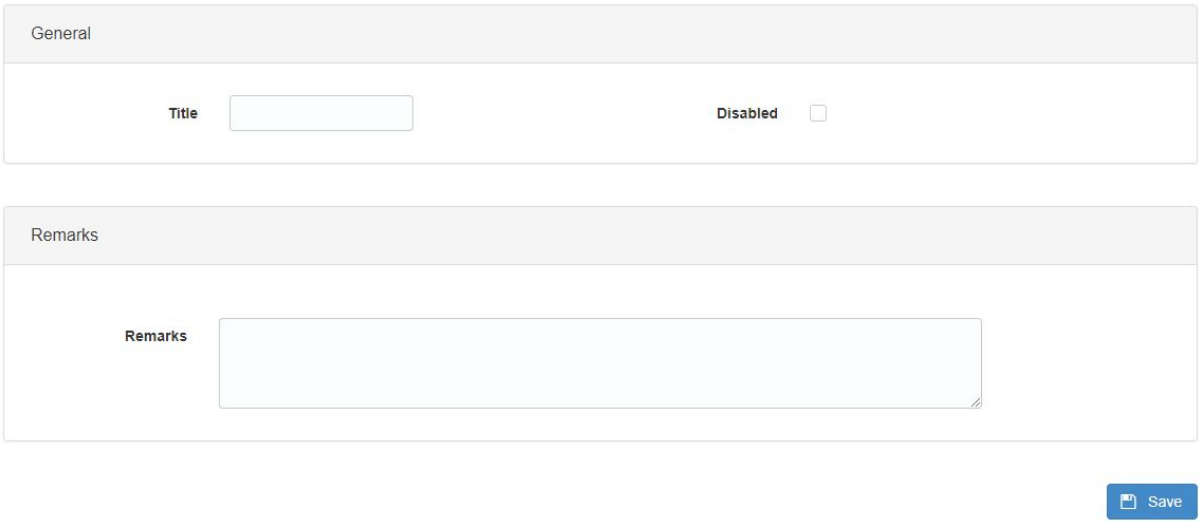

Figura 39: Vista per afegir Template

Per començar a la vista cal utilitzar el model declarat abans per poder utilitzar-lo a la vista. Això es fa posat: **@model Agrifood.CloudFP.WebApp.Models.TaskTemplateModel**

Tot seguit comença el formulari de dades de la pàgina fent servir codi c#.

```
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
```

Figura 40: Snippet 28: Declaració del formulari en la vista de detalls

Aquesta vista presenta a l'usuari un formulari a emplenar. aquest formulari es declara en la mateixa vista i s'imposa una mesura de seguretat per no permetre atacs externs. en quant al disseny del formulari s'utlitza un de semblant en tota l'aplicació i s'adapta al tamany del dispositiu de l'usuari.

El disseny utilitza un panell per mostrar una secció i cada secció divideix en 2 columnes els seus camps definint per cada camp una etiqueta al costat del camp associat.

El .Net MVC proporciona unes eines anomenades HtmlHelpers, que ajuden a generar codi html en la vista de forma intuïtiva i rapida.

@Html.HiddenFor(model => model.TaskTemplateId) serveix per guardar el valor de l'element però no mostrar-lo a l'usuari. En canvi el següent codi:

@Html.LabelFor(model => model.Description, TextResources.Description, null)

@Html.EditorFor(model => model.Description, new { htmlattributes = new {@class = "form-control"}})

genera el codi html següent:

```
<label for="Description">Description:</label>
<input type="text" class="form-control" id="Description" name="Description" value>
```

Figura 41: Codi html generat amb HtmlHelpers

Aquest codi esta vinculat al formulari i el valor introduït serà el valor que s'envia al controlador.

@Html.TextAreaFor(model => model.Remarks, 4, 20, new{ @class = "textareaclass"})

És el codi de l'ultim control que genera un control més gran per poder escriure més.

III. La vista per editar

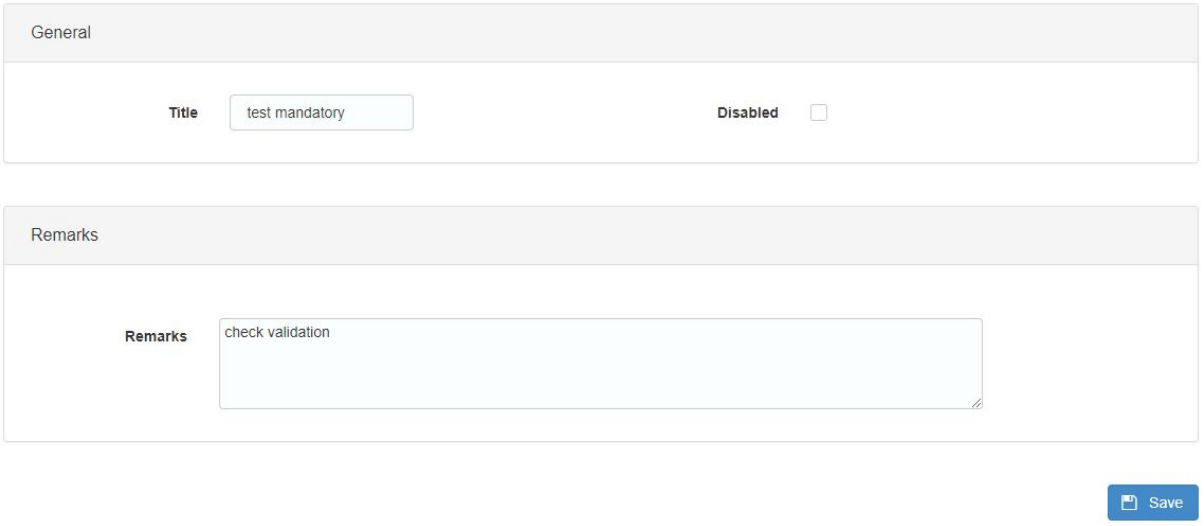

Figura 42: Vista per editar Template

La vista d'edició te la mateixa estructura que la de creació.

```
<div class="form-horizontal form-widgets col-sm-6">

  @Html.ValidationSummary(true, "", new { @class = "text-danger" })

  <div class="form-group">
    @Html.LabelFor(model => model.IsDisabled, Agrifood.CloudFP.WebApp.Resources.TextResources.
    <div class="col-sm-8 col-md-6" style="padding-top:7px">
      @Html.Kendo().CheckBoxFor(model => model.IsDisabled)
    </div>
  </div>

</div>
```

Figura 43: Snippet 29: Component de Telerik basat en HtmlHelpers

Apart de taules Telerik MVC també proporciona altres elements com captcha, gràfiques, i altres. Per mantenir l'estil de l'aplicació s'ha optat utilitzar aquests elements on es veuen millor que els per defecte, els proporcionats per Bootstrap o .Net MVC.

En les 2 vistes al pitjar sobre el boto blau s'agafen les dades del formulari i s'envien cap a la funció corresponent del controlador per ser tractades.

Tasca 1.2.1: Afegir camps a una Plantilla

Aquesta tasca és una continuació de la part de les dos anteriors. L'objectiu és aconseguir guardar diferents camps sota una mateixa plantilla.

El model

Cal afegir un model per guardar aquestes dades ja que no son ni les de plantilles ni les de camps són d'un altre taula que les relaciona.

```
23 references
public class TaskTemplateItemModel
{
    3 references | 0 exceptions, ? live
    public int TaskTemplateId { get; set; }

    [UIHint("FieldIdEditor")]
    6 references | 0 exceptions, ? live
    public int FieldId { get; set; }

    1 reference | 0 exceptions, ? live
    public string FieldDescription { get; set; }

    5 references | 0 exceptions, ? live
    public int SequenceNumber { get; set; }

    5 references | 0 exceptions, ? live
    public bool IsMandatory { get; set; }
```

Figura 44: Model de TemplateItem

El **TaskTemplateId i FieldId** s'utilitzen com identificadors d'aquesta classe.

El **FieldDescription** és el nom de l'element de la plantilla.

El **SequenceNumber** és un index segons el qual s'organitzen els elements de la plantilla.

El **IsMandatory** és una variable booleana que indica si cal completar el camp o no.

El controlador

El controlador d'aquesta entitat implementa les mateixes funcions que les 2 anteriors. Sense repetir els detalls de les funcions ja mencionades l'extructura del controlador queda de la següent forma:

```
[AuthRequired]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 0 requests, ? live | 0 exceptions, ? live
public ActionResult Items(int TemplateId, string Template)...

[AuthRequired]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 0 requests, ? live | 0 exceptions, ? live
public async Task<ActionResult> ReadTaskTemplateItems([DataSourceRequest]DataSourceRequest request, int TemplateId)...

[AuthRequired]
[AcceptVerbs(HttpVerbs.Post)]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 0 requests, ? live | 0 exceptions, ? live
public ActionResult AddTaskTemplateItem([DataSourceRequest] DataSourceRequest request, Models.TaskTemplateItemModel NewItem)...

//JSON
[AuthRequired]
[AcceptVerbs(HttpVerbs.Post)]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 0 requests, ? live | 0 exceptions, ? live
public ActionResult EditTaskTemplateItem([DataSourceRequest] DataSourceRequest request, Models.TaskTemplateItemModel Item)...

//JSON
[AuthRequired]
[AcceptVerbs(HttpVerbs.Post)]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 0 requests, ? live | 0 exceptions, ? live
public ActionResult DeleteTaskTemplateItem([DataSourceRequest] DataSourceRequest request, Models.TaskTemplateItemModel Item)...
```

Figura 45: Snippet 30: Funcions del controlador de TemplateItem

La vista

La vista de TemplateItem segueix el mateix model que Field i permet la creació i edició d'elements des de la mateixa vista. Això fa que la assignació de camps a la plantilla es faci de forma més rapida i entenedora. Utilitzant una nova pantalla cansaria l'usuari ja cada cop que vol afegir un nou camp hauria de tornar a la taula.

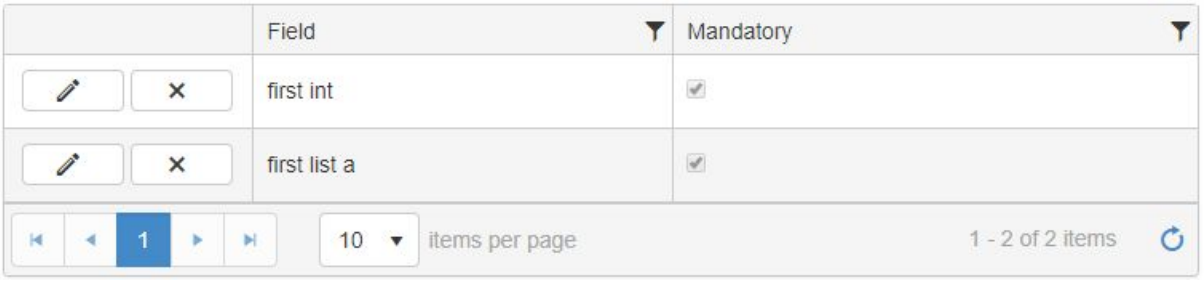

Figura 46: Vista llista de de TemplateItem

```
@(Html.Kendo().DropDownList()
        .Name("FieldId")
        .DataTextField("Description")
        .DataValueField("FieldId")
        .DataSource(source =>
        {
            source.Read(read =>
            {
                read.Action("GetFieldsDropdownList", "Fields");
            })
            .ServerFiltering(true);
        })
        .AutoBind(true)
)
```

Figura 47: Snippet 31: Codi de la vista per editar el camp de TemplateItem

Per poder editar els elements cal declarar una vista personalitzada i a més implementar al controlador la funció d'obtenir els tots els camps. En aquest punt sembla que s'està implementant un altre cop la funció de ReadFields() però no ja que aquest funció retorna només el ID i la descripció dels camps i serveix només per l'editor. Aquest és el codi principal que diferencia les 2 funcions:

return Json(fvm.Select(p=> new {FieldId = p.FieldId, Description = p.Description }), JsonRequestBehavior.AllowGet);

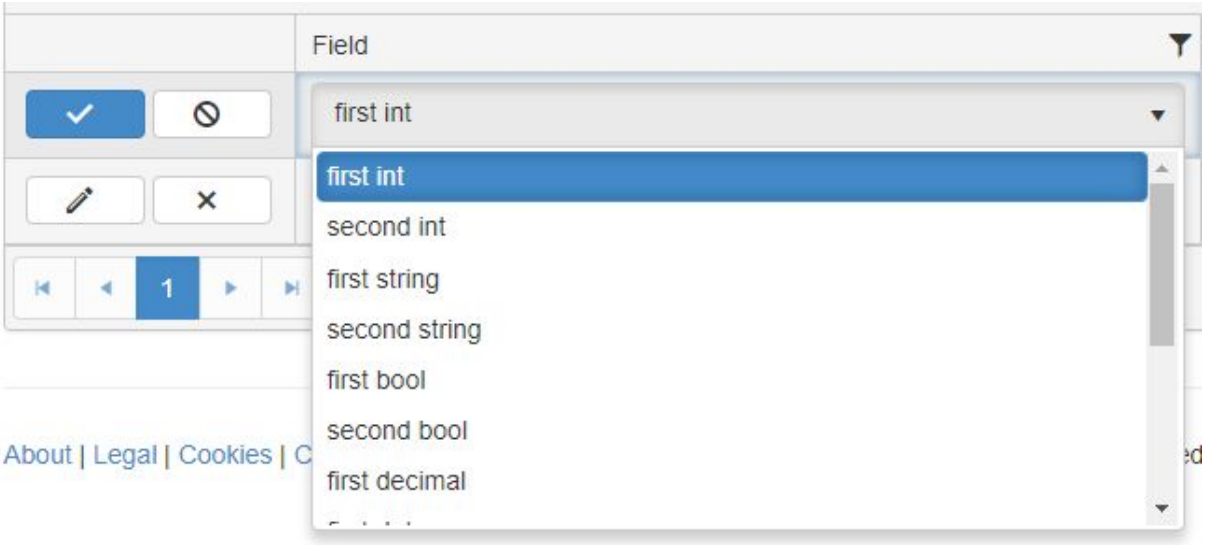

Figura 48: Vista resultant editant el camp de TemplateItem

Aquesta vista parcial només es crida a l'hora d'editar el Camp i presenta a l'usuari una llista amb tots els camps registrat a l'aplicació dels quals triarà un.

Tasca 1.3.0: Entitat Tasca V0

Aquesta entitat és el nucli del projecte i incorpora les altres entitats en el seu funcionament. Esta relacionada amb moltes altres entitats i va portar més temps implementar.

El model

Ja que la tasca te molts camps cal distingir entre els obligatoris i els secundaris. Els obligatoris porten una anotació per remarcar-los i a la hora de gestionar el formulari aquests camps obligaran a l'usuari a emplenar-los abans de poder guardar els canvis.

El **TaskId** és l'identificador d'aquesta classe.

El **ParentTaskId** és el valor identificador d'una tasca relacionada o tasca pare.

El **FarmId** és un camp obligatori i identifica la granja on s'ha de dur a terme la tasca.

El **Subject** és un camp obligatori i és el títol o nom de la tasca.

El **TaskTemplateId** és l'identificador de la plantilla d'aquesta tasca.

El **FarmId** és un camp obligatori i identifica la granja on s'ha de dur a terme la tasca.

El **GroupId** és l'identificador del grup d'animals dins de la granja relacionat amb la tasca.

El **AnimalId** és l'identificador de l'animal relacionat amb la tasca.

El **Status** és un marcador del progrés de la tasca, marcant si s'ha creat, acabat, completat o altres possibles estats.

La **StartDate** és la data d'inici de la tasca, amb data i hora.

La **EndDate** és la data de fi de la tasca, amb data i hora.

El **Description** és la descripció de la tasca, més llarga que el títol pot incloure informació útil per l'usuari assignat.

El **OwnerId** és l'identificador de l'usuari que ha creat la tasca.

El **AssignedUserId** és l'identificador de l'usuari assignat a la tasca que l'haura de completar.

El controlador

Aquesta part va ser potser la més senzilla dins de tota l'entitat ja que totes les operacions consistien en consultar el servei i validar el model com a les classes anteriors.

```
[AuthRequired]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 2 requests, ? live | 0 exceptions, ? live
public ActionResult Index()...

[AuthRequired]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 2 requests, ? live | 0 exceptions, ? live
public async Task<ActionResult> ReadTasks([DataSourceRequest]DataSourceRequest request)...

[AuthRequired]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 0 requests, ? live | 0 exceptions, ? live
public async Task<ActionResult> Details(string TaskId, string ErrorMessage)...

[AuthRequired]
[HttpPost]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 0 requests, ? live | 0 exceptions, ? live
public ActionResult Details(TaskViewModel model)...

[AuthRequired]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 1 request, ? live | 0 exceptions, ? live
public ActionResult AddNewTask(int TemplateId, string ParentTaskId, string ErrorMessage)...

[AuthRequired]
[HttpPost]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 1 request, ? live | 0 exceptions, ? live
public ActionResult AddNewTask(TaskViewModel model)...
```

Figura 49: Snippet 32: Funcions del controlador de Task

El controlador esta pensat per utilitzar la mateixa estructura que Templates: una vista de llistat amb les opcions d'esborrar[14] i carregar, una vista per afegir i una per editar els detalls que en aquest cas serveix per veure els camps de la plantilla afegida.

La vista

Aquestes vistes segueixen la mateixa estructura amb petits canvis.

Figura 50: Vista llista de Task

Com que la tasca conte tantes dades cal mostrar les més importants al llistat, tot i que estiguin buides.

Per la part d'edició i creació s'utilitzen diversos panells per dividir la informació de la tasca, la descripció i els elements de la tasca.

[14] tot i que no surt a la figura esta implementada també.

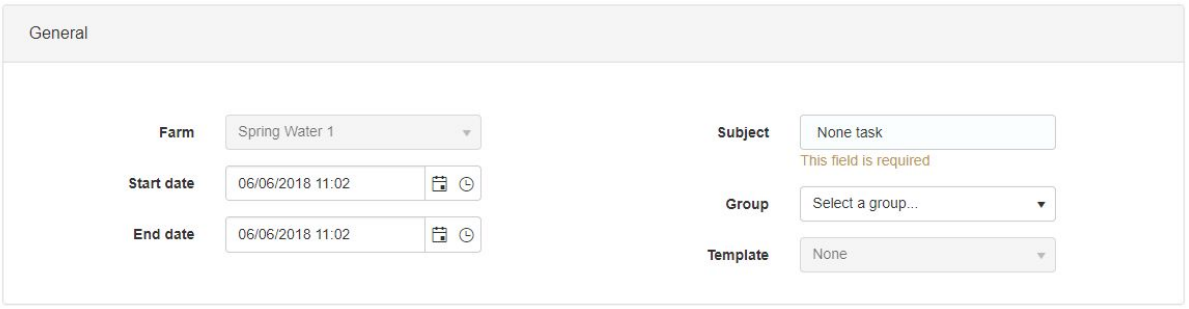

Figura 51: Part de la vista d'afegir de Task

Tots els elements de selecció com granja , grup i plantilla utilitzen una funció independent de cada que retorna tot el llistat i l'usuari selecciona la que vol; A l'hora de guardar només se passa el ID d'aquestes propietats de la mateixa forma que es fa a TemplteItems amb els Camps. El fet de que camps com FarmId i Subject que són obligatoris i estan marcats com tal al model fa que MVC .Net generi el a la capcelera html dels elements la etiqueta required[15].

I és en Aquest punt que s'acaba el primer esprint. Havent implementat les classes principals l'equip s'adona de que l'esquema inicial no és adient a les necessitats de FPW i FPM. Cal modificar la base de dades, actualitzar les classes ja creades i finalitzar la part de TaskItems(elements de la tasca). A més cal testejar i garantir el bon funcionament multiplataforma de l'aplicació, però tot això al segon esprint.

[15] required en html indica que l'element no pot estar buit a l' enviar el formulari.

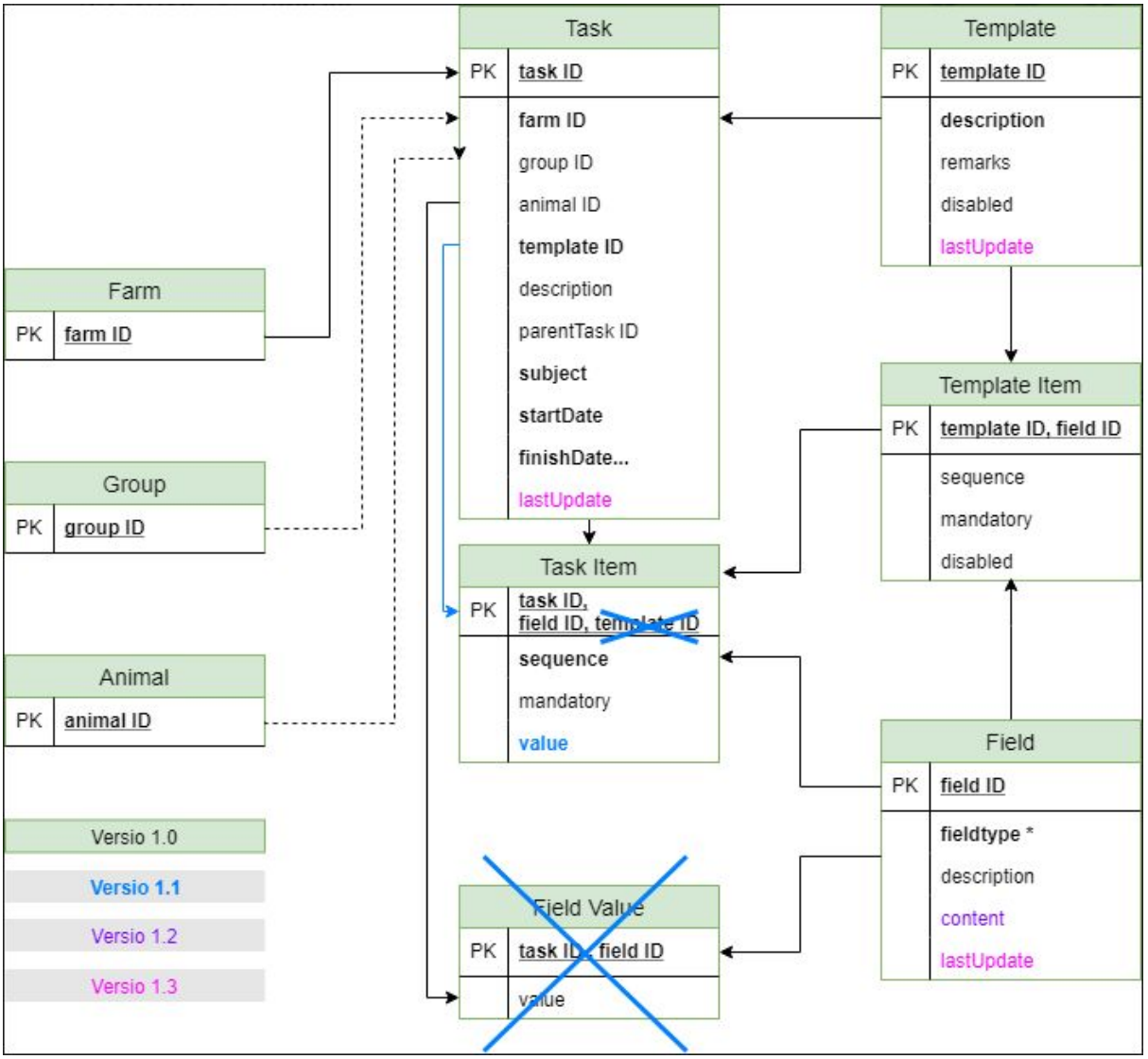

Figura 52: Esquema UML amb els canvis a la base de dades

Tal com estava pensada la base de dades l'aplicació guardava el valors dels camps en una taula que potser no crees al mateix temps que la tasca.Per aquest motiu és va proposar el primer canvi a l'estructura del darrere de l'aplicació: eliminar la taula de valors i guardar el valor dins dels elements de la tasca que si que es crearan quan es crea la tasca. A més qualsevol canvi dels camps pot implicar un canvi d'estat de la tasca a que pertanyen.

El següent canvi va ser afegir un nou camp a la taula de Field, on es guardaran dades addicionals.

Per ultim cal esmentar l'adició d'una data de l'ultima modificació que tot i que no implica grans canvis a FPW serveix per a que FPM es pugui actualitzar les tasques. L'aplicació mòbil utilitza una base de dades interna i no consulta constantment el servidor per mantenir-se actualitzat ja que suposaria un gran consum pel mòbil. La solució es utilitzar consultes periòdiques i notificacions per garantir un estat consistent de les 2 aplicacions.

Esprint 2

Com a resultat de la revisió i l'actualització del sistema s'han establert les següents tasques :

Tasca	Funcionalitat	Estimació	Dependències
aplicar canvis a les classes que depenen de "camp"	● afegir els camps de tipus llista.	4 hores	els canvis a la Base de Dades
depurar la classe "tasca"	● tractar tasca i els seus elements com un conjunt per crear i editar	12 hores	la interfície de l'aplicació mòbil, ja que ha de proporcionar els mateixos tipus de controls que la web
afegir un calendari per veure les tasques	● afegir un calendari mostrant una vista diària i mensual amb les tasques ● editar o veure els detalls de la tasca seleccionant-la	8 hores	una tasca depèn de la "plantilla" assignada que fa que tingui 0 o diversos "camps".

Figura 53: Taula 2: Backlog de tasques de l'esprint 2

Tasca 2.1: Entitat Camp V1

Com que es vol guardar un nou tipus de camp s'afegeix a l'Enum i aplicar els canvis pertinents a cada part del MVC.

```
16 references
public enum FieldType
{
    Integer = 1,
    String = 2,
    Boolean = 3,
    Decimal = 4,
    DateTime = 5,
    List = 6,
    Object = 7
}
```

Figura 54: Enum de FieldType versió nova

Aquest nou tipus representa una llista d'elements dels quals l'usuari podrà escollir un. La llista d'opcions s'ha de guardar al crear el camp i el valor quan es completa la tasca. Aquest camp pot tindre més usos en el futur però per ara només serveix amb aquest camp.

El model

Per al model de Camp s'afegeix un not atribut Content de tipus cadena de text on es guardaran les diferents opcions de la llista separades per coma. Com que al servei es retorna una cadena de string[16] cal transforma aquesta cadena tan quan el model rep com quan envia dades.

Content = string.Join(",", x.content) és el codi per transformar del contracte cap al model, x sent el contracte.

Content = (x.Content == null) ? new string[]{""} : x.Content.Split(",") és el codi per la funció de parseig de model cap al contracte on x és el model.

La vista

Ja que els canvis no afecten a la funcionalitat nomes a les dades podem saltar el controlador.

Sobre la vista cal dir només que s'ha afegit la columna per Content a la taula i com es una cadena de text l'editor de text per defecte funciona.

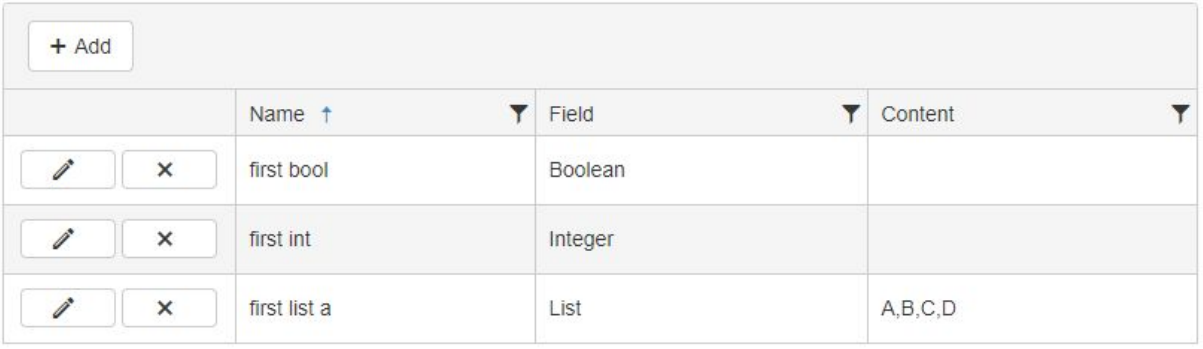

Figura 55: Vista llista de Camps actualitzada

Tasca 2.2: Entitat Tasca V1

Per la part de tasques cal redissenyar molta part de l'implementació, començant pel model.

El model

El primer canvi que cal fer és definir be el Task Item, ja que s'utilitzara de forma atòmica amb Task.

Aquest model contindrà els següents camps:

El **TaskId i FieldId** serveixen per identificar aquesta classe.

El **SequenceNumber** s'hereda del Template Item, serveix per ordenar.

El **IsMandatory** s'hereta del Template Item, serveix per marcar els camps importants.

El **FieldDescription** és el nom del Camp.

[16] string es el tipus de cadena de text en la majoria de llenguatges de programació

El **FieldType** s'hereta de Field i és el tipus del Camp.

El **Content** s'hereta de Field i és el contingut addicional del camp.

El **FieldValue** és on es guarda el valor del Camp, serà un string ja que es fàcil de parsejar.

```
switch (x.fieldType)
{
  case Common.CloudFPService.ServiceEnumsFieldType.Integer:
    tim.FieldValue = x.ValueInt.ToString();
    break;
  case Common.CloudFPService.ServiceEnumsFieldType.String:
    tim.FieldValue = x.ValueString;
    break;
  case Common.CloudFPService.ServiceEnumsFieldType.Boolean:
    tim.FieldValue = x.ValueBool.ToString();
    break;
  case Common.CloudFPService.ServiceEnumsFieldType.Decimal:
    tim.FieldValue = x.ValueDecimal.ToString();
    break;
  case Common.CloudFPService.ServiceEnumsFieldType.DateTime:
    tim.FieldValue = x.ValueDateTime.ToString();
    break;
  case Common.CloudFPService.ServiceEnumsFieldType.List:
    tim.FieldValue = string.Join(",",x.ValueList.ToArray());
    break;
  case Common.CloudFPService.ServiceEnumsFieldType.Object:
    tim.FieldValue = x.ValueObject.ToString();
    break;
  default:
    break;
}
```

Figura 56: Snippet 33: Tractament del valor en funció del tipus

A la hora de tracta amb el contracte cal mirar quin valor s'ha passat. El servei guarda els valors d'una forma diferent per cada abans de guardar-los a la base de dades. Aquest pas afegeix més seguretat a les operacions implicades. El mateix és fa en sentit contrari parsejant de string cap al tipus seleccionat quan s'envien dades cap al servei.

En quan a tasques cal afegir un nou element al model: una llista del model que acaben de crear. Això farà que les 2 entitats es comportin com una sola.

El controlador

Cal redefinir les comandes per afegir i editar de Task ja que ara impliquen també veure i modificar els Task Items.

A les funcions Get d'afegir i modificar cal recollir més informació del servei per mostrar una Tasca.

```
Models.TaskViewModel Details = new Models.TaskViewModel()
{
  TaskId = Guid.NewGuid().ToString(),
  OwnerId = userID,
  FarmId = (farmID == 0) ? 0 : farmID,
  GroupId = 0,
  AnimalId = 0,
  StartDate = new DateTime(DateTime.Now.Year, DateTime.Now.Month, DateTime.Now.Day + 1, 0,0,0),
  EndDate = new DateTime(DateTime.Now.Year, DateTime.Now.Month, DateTime.Now.Day + 1, 0, 0, 0),
  AssignedUserId = -1,
  TaskTemplateId = TemplateId,
  Status = (int)AppGlobals.TaskStatus.Created,
  ParentTaskId = ParentTaskId ?? " ",
  TaskItems = new List<TaskItemViewModel>()
};
```

Figura 57: Snippet 34: Creació del model per la vista d'edició i creació de Task

Després cal consultar el servei i capturar els Template Items de la plantilla que l'usuari ha seleccionat.

```
foreach (var item in templateItems)
{
  var tmp = con.GetField(item.FieldId, token, AppGlobals.APPLICATION_CODE);
  string tmpVal = " ";

  TaskItemViewModel tvm = new TaskItemViewModel()
  {
    FieldDescription = tmp.Description,
    FieldId = tmp.FieldId,
    FieldType = (Agrifood.CloudFP.WebApp.Classes.AppGlobals.FieldType)tmp.FieldType,
    FieldValue = tmpVal,
    Content = tmp.Content,
    IsMandatory = item.IsMandatory,
    SequenceNumber = item.SequenceNumber,
    TaskId = Details.TaskId,
  };
  Details.TaskItems.Add(tvm);
```

Figura 58: Snippet 35: Creació dels camps de la tasca per la vista de creació de Task

Tot seguit i per cada camp trobat cal crear un element de la tasca corresponent, assignant el tipus, l'ordre i si és important. En el cas d'afegir el Task Item tindrà un valor buit però en el cas de modificar es guarda el valor del camp ja existent ja que no utilitza els elements de la plantilla sinó que directament consulta els Task Items.

Tot aquest esforç serveix a la hora de guardar les dades (la funció POST) ja que no fa falta configurar res. el servei te implementada una funció que guarda tota la entitat amb els seus elements.

Cal Mencionar el Guid[17], que s'utilitza desde la web per identifica la tasca a la hora de guardar ja es la mateixa web que crea els elements associats a la tasca i que han de tenir el mateix identificador que la tasca.

```
con.UpdateTask((Agrifood.CloudFP.Common.CloudFPService.TasksContract)model, token, AppGlobals.APPLICATION_CODE);
foreach (var item in model.TaskItems)
{
    con.UpdateTaskItem(item, token, AppGlobals.APPLICATION_CODE);
}
```

Figura 59: Snippet 36: Codi per actualitzar la tasca i tots els camps

A la hora de guardar canvis a la tasca cal primer actualitzar la tasca i després actualitzar cada un del seus camps. I en quan la funció d'esborrar, només s'executa si la plantilla no te cap element.

La vista

La part més important de les vistes és la part corresponent a la llista de Camps.

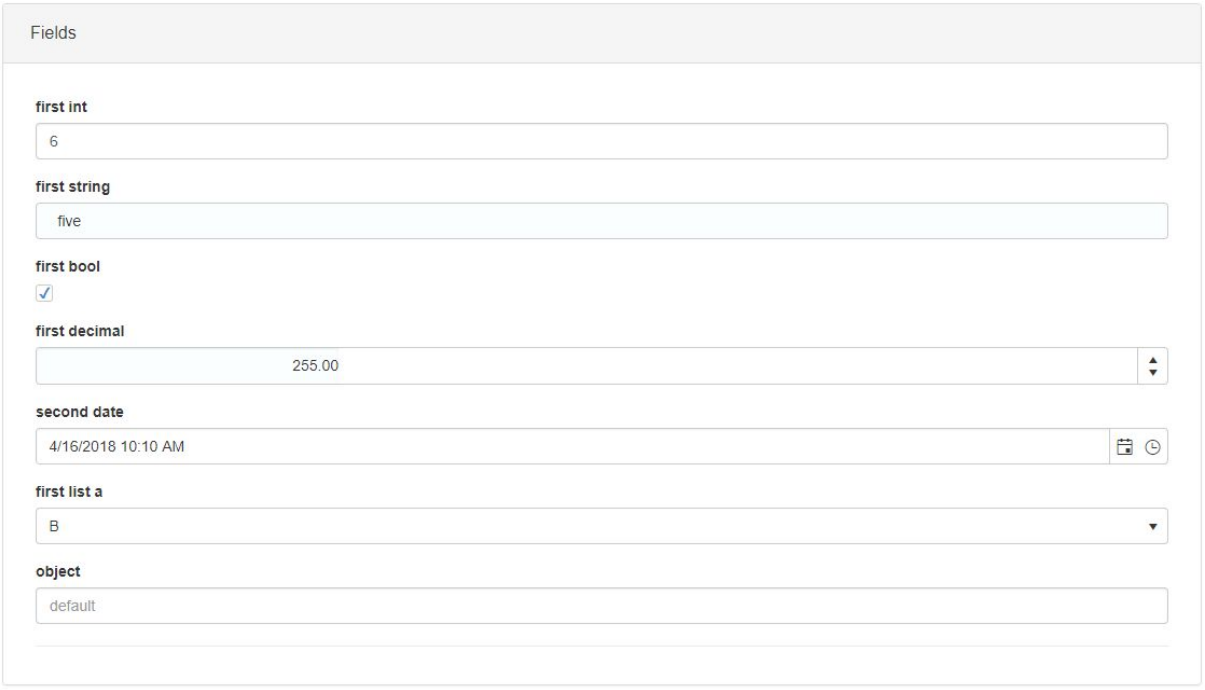

Figura 60: Vista dels diferents tipus de camps amb els seus editors

```
@if (Model.TaskItems.Count() == 0)
{
  <div class="form-group">
        <label >@Agrifood.CloudFP.WebApp.Resources.TextResources.Task_ErrorNoFields_Text</label>
  </div>
}
else
{
```

Figura 61: Snippet 37: Verificar si la tasca conté elements TaskItem

Per implementar aquest tros cal combinar molt c# i html. Primer cal mirrar si la llista conte elements.

[17] Guid (global unique identifier) és un nombre identificador generat pel sistema

El cas simple és que no conte elements i ho notifiquem a l'usuari mitjançant una etiqueta.

```
for (int i = 0; i < Model.TaskItems.Count(); i++)
{
  @Html.HiddenFor(m => m.TaskItems[i].TaskId)
  @Html.HiddenFor(m => m.TaskItems[i].SequenceNumber)
  @Html.HiddenFor(m => m.TaskItems[i].FieldType)
  @Html.HiddenFor(m => m.TaskItems[i].IsMandatory)
  @Html.HiddenFor(m => m.TaskItems[i].Content)
  @Html.HiddenFor(m => m.TaskItems[i].FieldDescription)
  @Html.HiddenFor(m => m.TaskItems[i].FieldId)
```

Figura 62: Snippet 38: Codi per generar els controls amagats de cada element TaskItem

En el cas contrari cal afegir tots Els Task Items al formulari. Per a que el controlador identifiqui que aquests camps son del tipus TaskItemModel cal afegir tots els membres de la classe, la majoria amagats. Peró amb això no hi ha prou per guardar el valor introduït per l'usuari

```
case Agrifood.CloudFP.WebApp.Classes.AppGlobals.FieldType.String:
  {
    <div>
      @(Html.Kendo().TextBox()
            .Name("TaskItems[" + @i + "].FieldValue")
            .Value(Model.TaskItems[i].FieldValue)
            .HtmlAttributes(new { style = "max-width:100%; width:100%", required = "required",
        )
    </div>
    break;
  }
```

Figura 63: Snippet 39: codi exemple per generar un editor de text per TaskItem

Utilitzant la variable FieldType recorrent cada posiblitat és genera un control específic per cada tipus. A més cal distingir si els controls son obligatori o no, cosa que afegeix més comprovacions i variacions de control. En quan a l'ordre dels elements de la llista, s'ordenen per SequenceNumber des del controlador.

Cal anomenar també els controls amb el mateix format que les variables amagades per a que el formulari els interpreti com una llista d'elements. I cal establir validacions per camps com la data o el decimal que no accepten cadenes de text.

Al final el codi queda resumit en 7 if()s a dins de un switch() amb 7 casos a dins d'un bucle de n elements que també esta dins d'un if()/else().

Havent completat aquest tros cal dir que no és l'únic element de les vistes d'edició i creació.

A la vista del llista es tria la plantilla de forma que es puguin generar els Task Items corresponenets a dins de la següent vista, sigui afegir o modificar.

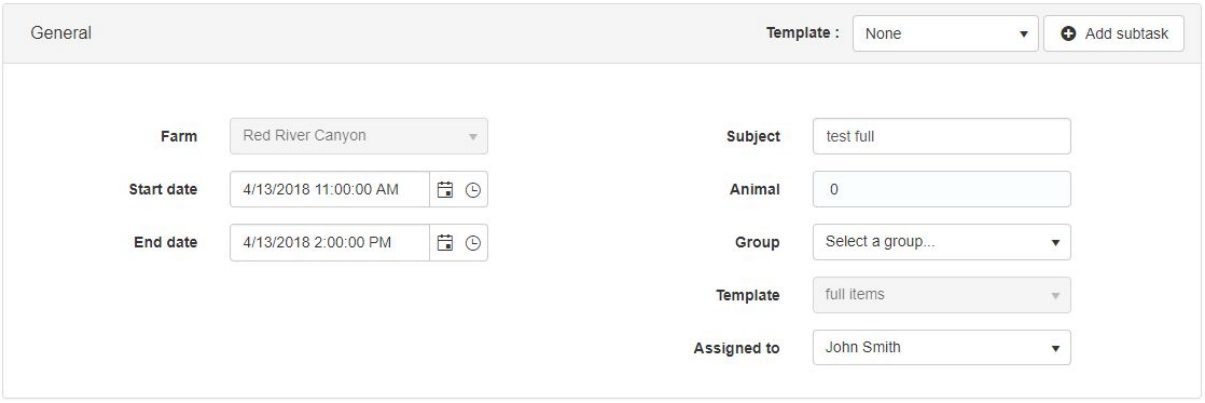

Figura 64: Part de la vista de detalls de Task actualitzada

L'assignació d'usuari es fa només a la modificació de la tasca, ja que això fa que la tasca canvii d'estat creat a assignat.

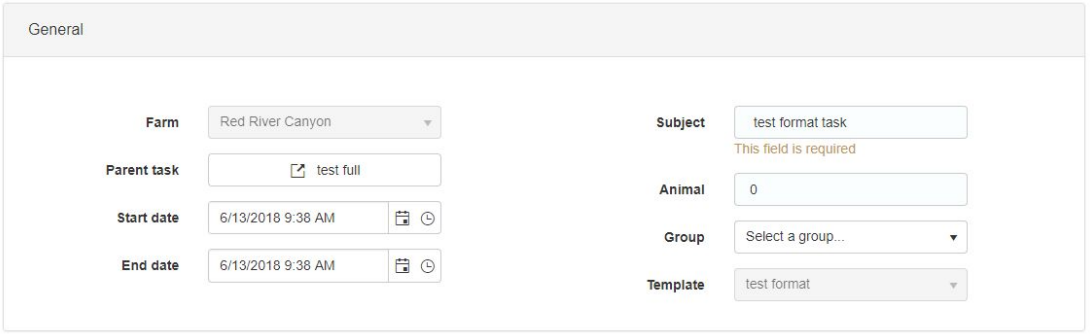

Figura 65: Part de la vista de creació amb tasca relacionada

A la vista d'afegir apareix un enllaç a la tasca relacionada, aquest camp correspon al ParentTaskId i només és mostra al crear una tasca a partit d'una altre existent i s'utilitza la mateix granja.

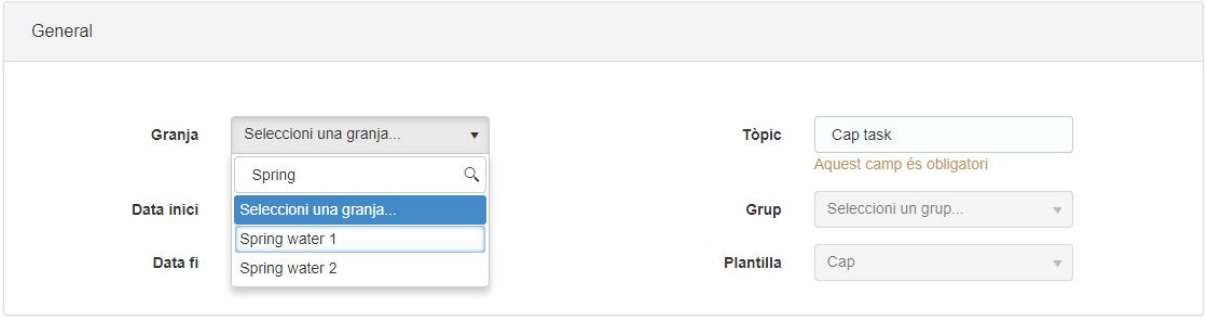

Figura 66: Control de selecció amb buscador per granges

A la vista d'afegir pel control de selecció de granja s'ha afegit un buscador bàsic. Aquest crida al controlador passant li el text introduït per l'usuari el el controlador li retorna només els noms de granja que contenen el text. El filtratge realitzat mitjançant Linq.

Ara cal parlar dels camps que encara no s'han esmentat però han portat més problemes en la solució FP en general: les dates.

```
function onStartChanged() {
  console.log(this.value());
  var endPicker = $("#EndDate").data("kendoDateTimePicker"),
    startDate = this.value();

  if (startDate) {
    startDate = new Date(startDate);
    startDate.setMinutes(startDate.getMinutes() + 30);
    endPicker.min(startDate);
    endPicker.value(startDate);
  }
}
```

Figura 67: Funció JS per actualitzar la data d'inici

Primer cal afegir-li un límits a la data d'inici ja que no pot ser després de la de fi. I cal definir que la data de fi sigui almenys la mateix que la d'inici. Tot això mitjançant un codi javascript que s'executa quan l'usuari canvia la data.

```
function onEndChanged() {
  var startPicker = $("#StartDate").data("kendoDateTimePicker"),
    endDate = this.value();

  if (endDate < startPicker.value()) {
    $("#dates_error").removeAttr('hidden')
  }
  else {
    $("#dates_error").attr('hidden', 'hidden')
  }

  if (endDate) {
    endDate = new Date(endDate);
    endDate.setMinutes(endDate.getMinutes() - 30);
    startPicker.max(endDate);
  }
}
```

Figura 68: Funció JS per actualitzar la data de fi

Després cal mostra un error si d'alguna forma l'usuari introdueix la data de fi inferior a la d'inici. A més quan l'usuari tria una data de fi valida aquesta serà el valor màxim que l'usuari pot triar com a data d'inici. I com que les dates són difícils de tractar cal afegir un codi al validador de Jquery per a que accepti dates en un format estàndard en funció de la cultura/idioma seleccionada.

Tasca 2.3: Vista Calendari V0

Havent completat la part de Tasca cal afegir una nova pantalla on s'ensenyaran les tasques del més o dia en curs. A partit d'aquest pantalla es poden accedir una tasca per visualitzar seleccionant-la.

El model

Com que utilitzem el component Scheduler de Telerik MVC el model utilitzat ha de complir amb la plantilla del component. Cal que el model implementi una interfície Kendo.MVC.UI.ISchedulerEvent.

Sona complicat però en realitat cal afegir quatre o cinc camps extra al model. Aquest són com els camps per defecte i tot i que no s'utilitzen per la tasca s'han de declarar. Són coses com:

El **Start** - la data d'inici.

El **End** - la data de fi.

El **StartTimezone** - la zona horària d'inici.

El **EndTimezone** - la zona horària de fi.

El **RecurrenceRule** - la regla de repetició si es que l'event es repeteix.

El **RecurrenceRuleException** - la excepció a la regla, si per exemple no es repeteix el diumenge.

El **Title** - el títol de l'event.

```
public static implicit operator SchedulerEventViewModel(Agrifood.CloudFP.Common.CloudFPService.TasksContract x)
{
  SchedulerEventViewModel svm = new SchedulerEventViewModel()
  {
    TaskId = x.TaskId,
    ParentTaskId = x.ParentTaskId,
    TaskTemplateId = x.TaskTemplateId,
    FarmId = x.FarmId,
    GroupId = x.GroupId,
    AnimalId = x.AnimalId,
    Subject = x.Subject,
    Description = x.Description,
    OwnerId = x.OwnerId,
    AssignedUserId = x.AssignedUserId,
    Title = x.Subject,
    IsAllDay = false,
    Start = x.StartDate ?? DateTime.MinValue,
    End = x.FinishDate ?? DateTime.MinValue,
    StartTimezone = null,
    EndTimezone = null,
    RecurrenceRule = null,
    RecurrenceException = null,
    Status = x.Status
  };

  return svm;
}
```

Figura 69: Model del Scheduler

Tot i això es pot utilitzar el mateix contracte que les tasques ja que seran el mateix objecte mostrat d'una forma diferent en un control més complex.

El controlador

Com que utilitzem el component Scheduler de Telerik MVC el model utilitzat ha de complir amb la plantilla del component.

```
[AuthRequired]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 4 requests, ? live | 0 exceptions, ? live
public ActionResult Schedule()...

[AuthRequired]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 4 requests, ? live | 0 exceptions, ? live
public async Task<ActionResult> ReadTasksForScheduler([DataSourceRequest]DataSourceRequest request)...

[AuthRequired]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 0 requests, ? live | 0 exceptions, ? live
public ActionResult DeleteTaskForScheduler([DataSourceRequest] DataSourceRequest request, Models.TaskViewModel Task)...

[AuthRequired]
[HttpPost]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 0 requests, ? live | 0 exceptions, ? live
public ActionResult AddNewTaskForScheduler([DataSourceRequest] DataSourceRequest request, SchedulerEventViewModel model)...

[AuthRequired]
[CheckComponent(ComponentName = AppGlobals.COMPONENT_FP_TASK)]
0 references | 0 requests, ? live | 0 exceptions, ? live
public ActionResult UpdateTaskForScheduler([DataSourceRequest] DataSourceRequest request, SchedulerEventViewModel model)...
```

Figura 70: Funcions del controlador pel Scheduler

La funció Index retorna la vista amb el component del calendari.

La funció de lectura retorna totes les tasques que apareixen al calendari, transformant el contracte de Task cap al nou model creat. La funció d'edició i la de creació fan trampa ja que treballen amb el model de Scheduler com ho exigeix el component de Telerik però a la hora de tractar les dades transformen l'event en una nova tasca i s'insereix o actualitza utilitzant els mateixos contractes de Task.

Per no tapar tota la vista de l'usuari i poder treballar de forma més neta els camps no apareixen en el formulari d'edició del calendari. L'usuari només tracta amb les dades principals de la tasca que un cop creada és pot editar de forma separa i en complet.

La funció d'esborrar es comporta igual que a Task i no deixa esborrar tasques assignades o en curs.

La vista

Com que utilitzem el component Scheduler de Telerik MVC el model utilitzat ha de complir amb la plantilla del component. Aquest component pot ocupar tota la part central de pantalla però presenta tot la informació i control de forma intuïtiva i clara a l'usuari.

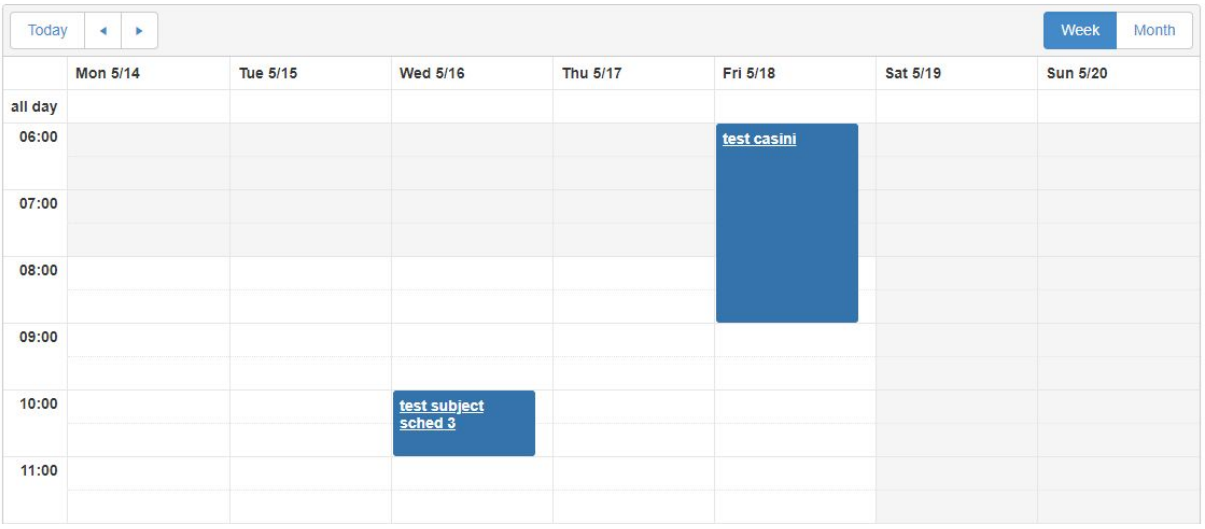

Figura 71: Vista amb el calendari mostrant tasques

```
@(Html.Kendo().Scheduler<Agrifood.CloudFP.WebApp.Models.SchedulerEventViewModel>()
    .Name("scheduler")
    .Date(DateTime.Now)
    .StartTime(6,0,0)
    .EndTime(22,0,0)
    .CurrentTimeMarker(c => c.UseLocalTimezone(true))
    .Views(views =>
    {
      views.WeekView(weekView => weekView.Title(Agrifood.CloudFP.WebApp.Resources.TextResources...
      views.MonthView(monthView => monthView.Title(Agrifood.CloudFP.WebApp.Resources.TextResources
    })
```

Figura 72: Snippet 40: Declaració del calendari

De la mateixa forma que es va declarà el grid cal declarà el scheduler. Cal configurar quines vistes apareixeran i marcar les opcions enfocades cap a l'usuari com per exemple mostrar la hora actual dins de la vista.

```
.EventTemplate(
  "<div class='event-template'>" +
      "<p>" +
      "<a href='Details/?TaskId=#= TaskId #'>" +
      "<u> #= Subject #</u>" +
      "</a>" +
      "</p>" +
  "</div>")
.DataSource(dataSource => dataSource
    .Model(model =>...)
  .Read(read => read.Action("ReadTasksForScheduler","Tasks"))
  .Destroy(destroy => destroy.Action("DeleteTaskForScheduler", "Tasks"))
  .Create(create => create.Action("AddNewTaskForScheduler", "Tasks"))
  .Update(update => update.Action("UpdateTaskForScheduler", "Tasks"))
)
.Events(e => e.Remove("changeEvent").Save("changeEvent").Edit("titleEdit"))
```

Figura 73: Snippet 41: Codi per la font de dades i la plantilla del calendari

Com que el que les tasques que apareixen al calendari no són els objectes de base de telerik s'utilitza una plantilla per mostrar el nom de la tasca vinculant cap a vista de detalls d'aquella tasca. D'aquesta forma l'usuari età a un pas de completar la tasca si la te assignada.

Si en canvi l'usuari vol afegir una tasca rapida ho pot fer ja que el component incorpora aquesta funcionalitat per defecte. Només cal crear una altre vista de plantilla que el calendari utilitzi per mostrar a l'usuari. Això és fa perquè molts dels camps per defecte no s'utilitzen per ara i donarien lloc a confusió si l'usuari els podes modificar sense que es guardessin.

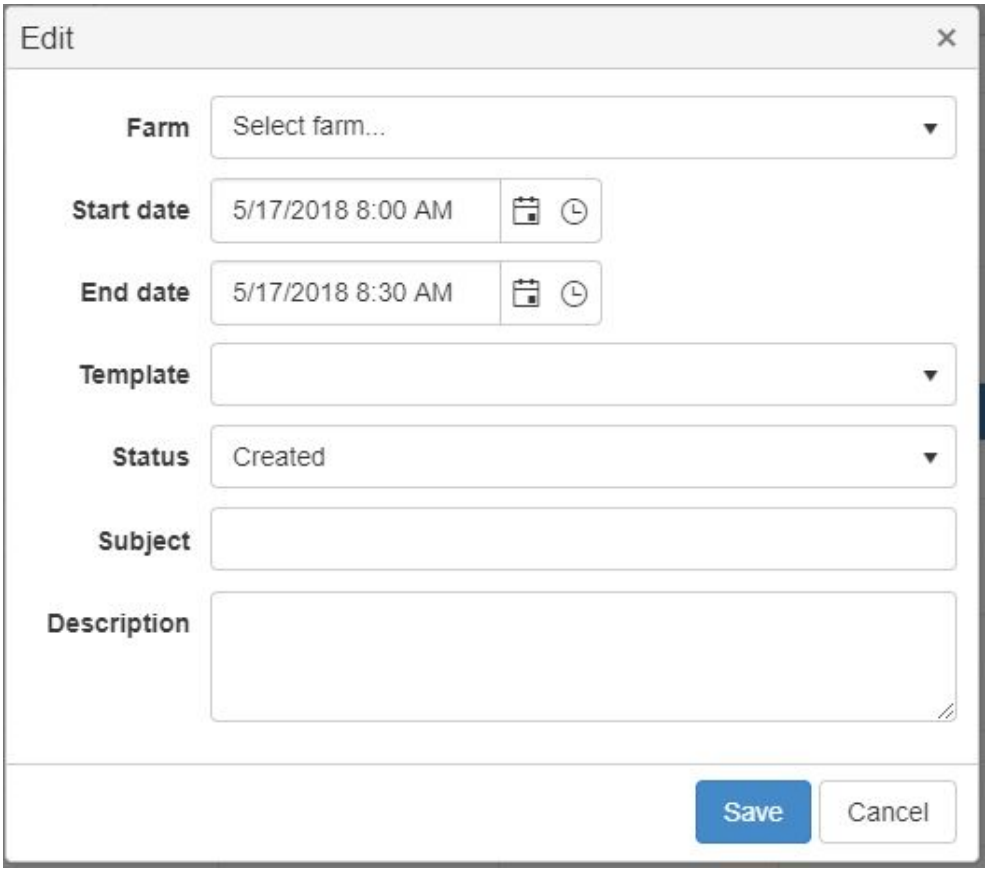

Figura 74: Modal per afegir la tasca al calendari

Aquesta vista mostra els camps més importants de la tasca i utilitza la mateixa funcionalitat que la vista de creació de tasca per garantir el mateix resultat.

Esprint 3

Aquest esprint es va realitzar per acabar el modul de tasques i definir el seu lloc dins de l'aplicació. Al ser un esprint curt d'una setmana es van definir només aquestes tasques:

Tasca	Funcionalitat	Estimació	Dependències
afegir la part de tasques dins del menú intern d'una granja	• poder afegir, eliminar, editar i llistar les tasques d'aquella granja	4 hores	la granja i les taques
afegir una vinculació entre plantilles i rols d'usuaris de la granja	• tractar tasca i els seus elements com un conjunt per crear i editar	8 hores	les plantilles i els rols definits

Figura 75: Taula 3: Backlog de tasques per a l'esprint 3

Tasca 3.1: Afegir Tasca dins de granges V1

Poc s'ha de dir d'aquest assumpte ja que és una copia de la Tasca 2.2 reutilitzant el model i replicant el controlador i vistes dins del controlador de granges.

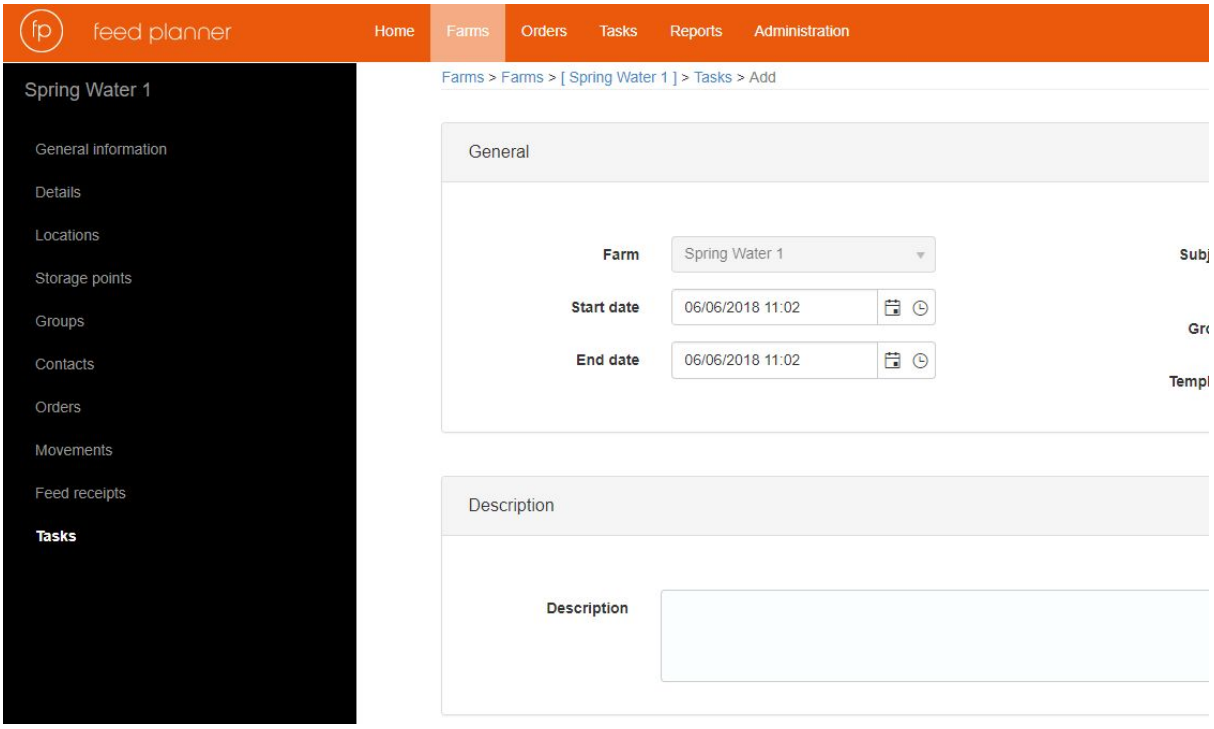

Figura 76: Vista de les tasques dins de la secció de detalls de granja

En aquest cas com que hi ha un granja seleccionada no és pot seleccionar cap altre.

Tasca 3.2: Entitat Plantilla V1

Cal afegir uns canvis a les plantilles ja que es vol donar una una experiència més rapida a l'usuari a l'hora de crear i gestionar les plantilles i conseqüentment les tasques. Per començar cal definir que els rols estan tipificats a la plataforma poden ser : veterinari, propietari, encarregat, visitador i més si és vol.

El model

Cal primer afegir una forma de verificar els rols als quals esta assignada la plantilla ja que només aquells usuaris la podran utilitzar per crear una tasca. Com que la plantilla no és una cosa exclusiva de una plantilla a la base de dades s'ha afegit una taula relacional per guardar els rols de cada tasca. Pel model de Template això implica crear dos nous membres:

public int [] Roles {get; set;} i **public int[] newRoles {get; set; }**

Aquests vectors guarden el ID dels rols. La raó per la qual s'utilitzen dos s'explica en la part del controlador.

El controlador

La primera funció en canvia és la d'afegir plantilles, la POST. Aquest funció rep un model com abans però ara després d'inserir la Template i rebre el ID cal afegir els rols escollits per l'usuari.

```
if (model.TaskTemplateId > 0)
{
    foreach (var item in model.Roles)
    {
        con.InsertFarmRoleTaskTemplate(item, model.TaskTemplateId, token, AppGlobals.APPLICATION_CODE);
    }
}
else
{
    return RedirectToAction("AddNewTemplate", "TaskTemplates");
}
```

Figura 77: Snippet 42: Codi per crear els rols de la plantilla al crear-la

El mateix no és pot fer per la funció d'edició ja que l'usuari pot afegir i eliminar rols si vol i aquest canvis no es veurien reflectits en un vector. Per això s'utilitza el segon vector que clona el primer en la funció GET d'edició i és el que es modifica per l'usuari en aquesta vista.

```
var toAdd = model.newRoles.Where(role => !model.Roles.Contains(role)).ToArray();
foreach (var item in toAdd)
{
    con.InsertFarmRoleTaskTemplate(item, model.TaskTemplateId, token, AppGlobals.APPLICATION_CODE);
}
var toDelete = model.Roles.Where(role => !model.newRoles.Contains(role)).ToArray();
foreach (var item in toDelete)
{
    con.DeleteFarmRoleTaskTemplate(item, model.TaskTemplateId, token, AppGlobals.APPLICATION_CODE);
}
```

Figura 78: Snippet 43: Codi per actualitzar els rols a l'actualitzar la plantilla

Per actualitzar la plantilla cal a més d'actualitzar la seva entitat mirar els 2 vectors de rols. Els rols nous que cal afegir estaran només en el newRoles i no en l'altre; En canvi si un rol no esta present en el segon vector però si que estava en Roles vol dir que s'ha eliminat i cal peticionar el servei que ho posi en practica.

Per acabar cal mencionar que la funció de lectura per mostrar el llistat també s'ha canviat ara utilitza un rol per mostrar només les plantilles assignades en aquest.

La vista

El canvis de la vistes són petits però importants.

Començant per la pantalla d'edició s'afegeix un control nou de Telerik anomenat multiselect que funciona de la mateixa forma que el dropdownlist però permet seleccionar i mostrar múltiples valors.

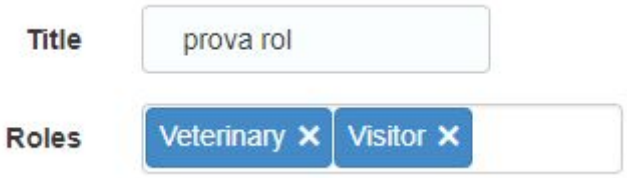

Figura 79: Control per seleccionar diversos rols

En la pantalla de creació aquest component esta vinculat amb la propietat Roles del model, però a la edició aquest control actua sobre newRoles; l'altre vector està vinculat a un control amagat de forma semblant al TaskTemplateId.

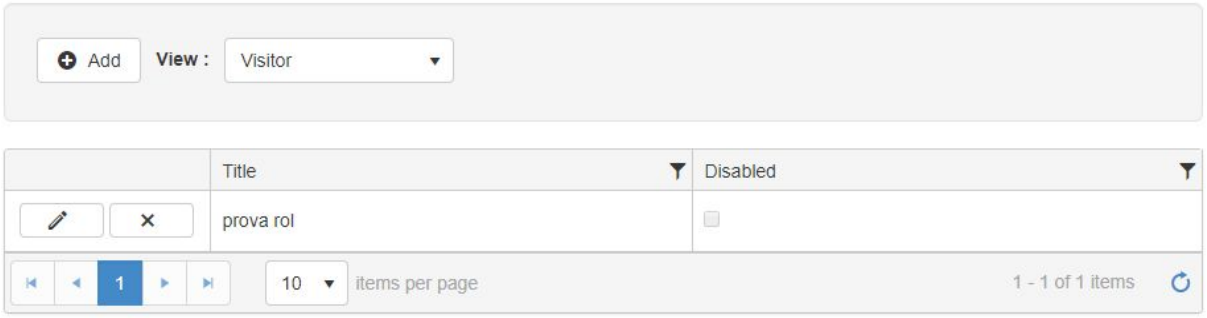

Figura 80: Vista llista de Template modificada i amb un filtre per rol

En la pantalla del llistat ja actualitzada sota un nova guia de disseny, s'afegeix un filtre per rol. Aquest control guarda el id del rol que l'usuari vol veure les plantilles.

Cal notificar el controlador de la opció que l'usuari ha seleccionat i com que el c# no es codi actiu en la pagina s'utlitza JS.

```
.Events(e =>e.Error("onDataSourceError"))
.Read(read => read.Action("ReadTemplatesFiltered", "TaskTemplates").Data("getParams"))
.Destroy(destroy => destroy.Action("DeleteTemplate", "TaskTemplates"))
```

Figura 81: Snippet 44: Codi afegit a la funció de lectura modificada

```
function getParams() {
    var farmrole = $("#FarmType").data("kendoDropDownList");
    var roleId = 1;
    var val = farmrole.value($("#value").val());
    var role = (val == "") ? roleId : val;
    return {
      FarmRoleId: role,
    }
}
```

Figura 82: Snippet 45: Funció JS per obtenir el rol seleccionat

Aquest funció agafa el valor del control si es que l'usuari ha seleccionat cap, sinó agafem el primer rol i el passem a la funció de lectura

Aquí s'acava la part de l'implementació i a continuació es detallen les conclusions del projecte.

6 Conclusions

Anàlisi de la planificació

Primer cal definir la planificació inicial a contrastar amb la realitat.

Fase	Inici	Fi
Anàlisis de la proposta	5/2/2018	18/2/2018
Disseny	19/2/2018	4/3/2018
Implementació inicial	5/3/2018	23/3/2018
Testeig inicial	2/4/2018	15/4/2018
Implementació final	16/4/2018	29/4/2018
Testeig final i aprovació	2/5/2018	16/5/2018

Figura 83: Taula 4: La planificació Inicial

El projecte va començar seguint el plan establert però poc a poc degut a una serie d'esdeveniments sota i fora del control de l'equip es va allargar. El disseny fet el primer dia va canviar almenys 5 vegades afectant indirectament a FPW i directament a l'aplicació FPM.

Un dels objectius era definir una experiència semblant tant a web com a mòbil i això es va aconseguir a base de reiterar sobre conceptes i detalls fins que tot funciones. Degut als canvis la fase de testeig va passar més com una fase continua de tot el projecte i va sol·licitar més hores de les previstes.

Tot i això el resultat és un bon producte, estable ampliable i de qualitat. aporta valor a l'aplicació i aporta nous coneixements a l'equip.

Al principi tot es va executar d'acord amb la planificació acabant el disseny i anàlisi en el primer mes. Peró la resta de fases van desbordar. La implementació inicial va tardar més de l'esperat ja que l'equip no tenia la mateixa idea al ment i es van trobar moltes coses a faltar i tenir que afegir i tornar a provar.

En quant al testeig potser fa falta més i el fet de treballar en dos aplicacions a l'hora va crear una serie de problemes que es van resoldre amb comunicació i molta calma. Semblava que cada cop que una part canviava l'altre deixes de funcionar en alguna forma.

Tot i així a la hora de l'ultim esprint s'havia construït una base estable i expansible sobre la qual es podia treballar i testejar be.

Conclusions personals

El projecte ha sigut una experiència enriquidora amb moments més bons que altres. L'esforç crec que ha valgut la pena i després de moltes revisions i afinaments puc dir amb cert orgull que aquest és el meu projecte , aquest és el meu codi; Podria ser millor, segurament i si continuo treballant amb .Net aprendre més i podré resoldre més problemes o evitar que surtin directament.

Aquest projecte m'ha ensenyat que el testeig és una part important del desenvolupament i que les pautes s'han de seguir a peu de lletra si és que hi ha cap pauta per al que estàs intentant fer. Per contrastar, fa falta iniciativa davant de problemes nous o complicats ja que una solució mig bona és millor que un usuari enfadat.

En quan al projecte en general no ha sigut tan difícil, l'equip de Caedis m'ha ajudat molt i m'han ensenyat a ser professional. El mateix no es pot dir pel que fa aquest informe, ha costat més que potser tota la implementació del projecte però en cap moment no he estat satisfet amb el resultat. Al fi i al cap tots som humans i hem de moure cap a davant sense arrossegar les mancances.

7 Treball futur

Ampliació

La primera possibilitat per madurar l'aplicació es afegir noves funcionalitats a la mateixa, com per exemple:

I. Afegir un panell amb informació general de les tasques com estadístiques, usuaris més actius o altres
II. Afegir un panell de configuració i personalitzar l'experiència de usuari afegint coses com prioritats, costos associats i potser notificacions
III. Expandir la part de rols per discriminar les tasques establin relacions entre els usuaris. Proporcionar camps addicionals en funció del rol assignat i més.

Integració

L'altre possibilitat és treballar per fer el modul més obert cap a altres aplicacions o components. Unes idees serien:

I. Integrar el modul de tasques amb un altre solució de l'empresa.
II. Sincronitzar les tasques amb un calendari extern com el de Google o Microsoft.

8 Bibliografia i Referències

https://stackoverflow.com/, 10/06/2018, web d'ajuda pels programadors

https://www.w3schools.com/, 10/06/2018, web amb exemples de programació web

https://www.versionone.com/agile-101/agile-methodologies/, 7/05/2018, informació sobre Agile

https://searchsoftwarequality.techtarget.com/definition/Scrum-sprint, 7/05/2018, informació Esprint

https://demos.telerik.com/aspnet-mvc/, 5/06/2018, controls de Telerik MVC

https://www.asp.net/mvc , 3/03/2018, informació .NET MVC

https://tortoisesvn.net/ , 4/03/2018, informació Tortoise SVN

https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/using-data-contracts, 1/03/2018 informació DataContracts

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/,1/04/2018, informació sobre Linq

https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-2.1,2/04/2018, informació sobre les vistes Razor